DEEP REPRESENTATION LEARNING ON HYPERSPHERE

A Dissertation
Presented to
The Academic Faculty

By

Weiyang Liu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Computer Science

Georgia Institute of Technology

August 2020

DEEP REPRESENTATION LEARNING ON HYPERSPHERE

Thesis committee:

Dr. Le Song, Advisor
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Bhiksha Raj
School of Computer Science
Carnegie Mellon University

Dr. James M. Rehg
School of Interactive Computing
Georgia Institute of Technology

Dr. Yao Xie
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Duen Horng Chau
School of Computational Science and Engineering
Georgia Institute of Technology

Date approved: July 15, 2020

Everything should be made as simple as possible, but not simpler.

Albert Einstein

To my parents and Rongmei

ACKNOWLEDGMENTS

I enjoy every step along the journal to pursue my Ph.D. degree at Georgia Tech. I am deeply thankful to many individuals who have contributed to the completion of this thesis in many different ways.

First of all, I feel strongly indebted to my thesis advisor Le Song. Without his supportive, visionary and insightful guidance, I could not achieve what I have now. It is his encouragement that helps me make up my mind to pursue a career in academia. I feel forever grateful to him.

I would like to express my deep gratitude to James M. Rehg. We have worked together in quite a few exciting projects and I have learned a lot from his outstanding expertise and profound research vision. It is my great honor to have worked with him.

I am also grateful to all of my thesis committee members: Duen Horng Chau, Bhiksha Raj, James M. Rehg, Le Song and Yao Xie. Their constructive suggestions to my thesis research are very important for me to finish my dissertation.

I am very fortunate to have worked with some of the best researchers in my field: Animashree Anandkumar, Yu Chen, Jan Kautz, Bhiksha Raj, Anshumali Shrivastava, Linda Smith, and Hongyuan Zha. The collaborations with them have prominent impact on my research and benefit me a lot. My research has also benefited significantly from interacting with a group of wonderful coauthors: Zhehui Chen, Beidi Chen, Bo Dai, Hanjun Dai, Chen Feng, Animesh Garg, Ahmad Humayun, Ming Li, Xingguo Li, Rongmei Lin, Zhen Liu, Albert Shaw, Feng Wang, Yisen Wang, Wei Wei, Yandong Wen, Li Xiong, Meng Yang, Zhiding Yu, Yan-Ming Zhang, and Tuo Zhao. Particularly, I have learned a lot from Bo Dai in all those profound discussions, and his research attitude has greatly inspired me.

I will never forget all the wonderful memories when working with Yandong Wen, Zhen Liu and Zhiding Yu. It has been almost 10 years since I first met Yandong. We are close friends and collaborators in the past decade, and He has significant influence on me. I can

still remember all those good old days when we discuss random research ideas and get inspired from each other. He is the one that encourages me to conduct research in deep learning back in 2015. Without him, I could not have achieved such a good start in deep learning research. Zhen is one of my closest collaborators and friends during my Ph.D. study in Georgia Tech. We always discuss ideas and share thoughts on recent research advances. We have done many interesting projects together and his enthusiasm on research is always an inspiration to me. It was a truly unforgettable experience working with him. Zhiding is one of the first people that taught me how to do impactful research. We have known each other since 2014, and it was his encouragement that gives me the confidence to apply for Ph.D. I have learned significantly from him in writing papers and conducting high-quality research.

I must thank all of my friends, labmates and colleagues: Binghong Chen, Xinshi Chen, Wenbo Chen, Nan Du, Shrivastava Harsh, Haoming Jiang, Elias Khalil, Shibo Li, Jiasen Lu, Zhaoyang Lv, Haoran Sun, Yichen Wang, Bo Xie, Jianwei Yang, Yuyu Zhang and many others, for every enjoyable moment.

Finally and most importantly, my family is always there for me no matter what happened in life. I feel forever indebted to my parents for the unconditional support and love. I would never have accomplished this without their selfless sacrifices. I am extremely grateful to my girlfriend, Rongmei, for her endless support and love. She is always the one that tells me to follow my heart and chase my dream. It is her constant company that makes my Ph.D. journey much easier and more enjoyable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

How to efficiently learn discriminative deep features is arguably one of the core problems in deep learning, since it can benefit a lot of downstream tasks such as image classification, object detection, etc. In this dissertation, we present a unified deep representation learning framework on hypersphere, which introduces a hyperspherical inductive bias into deep neural networks. We discuss our framework from four perspectives:

Learning objectives on hypersphere. To learn deep features that are discriminative on hypersphere, we propose a general framework to design learning objectives. As instances, we design the large-margin softmax loss and the angular softmax loss to learn features that have large inter-class angular margin. We show that replacing the standard loss function in neural networks with these large-margin loss functions leads to state-of-the-art performance on image classification and face recognition.

Neural architectures on hypersphere. To benefit the representation learning on hypersphere, we propose a hyperspherical neural network that makes each layer in the neural network to operate purely on angles. Furthermore, we generalize the hyperspherical neural network to the decoupled neural network that decouples the inner product into an angular activation and a norm activation. These new neural architectures are shown to have better generalization, faster convergence and stronger robustness against adversarial perturbations, compared to the standard neural networks.

Regularizations on hypersphere. Because the widely used weight decay regularization in neural networks is no longer effective in our hyperspherical learning framework, it is necessary to use alternative regularizations to regularize the direction of neurons rather than the norm of neurons. To this end, we propose the minimum hyperspherical energy regularization that minimizes a potential energy defined on hypersphere in order to encourage the hyperspherical diversity of neurons. Moreover, we further propose the compressive hyperspherical energy that is easier to minimize than the standard hyperspherical energy.

Both regularizations show superior empirical generalization and convergence performance on many supervised learning tasks.

Hyperspherical Training paradigm. Inspired by the observation that minimizing hyperspherical energy of neurons leads to better generalization, we propose a generic framework – orthogonal over-parameterized training that can provably minimize the hyperspherical energy in a principled way instead of using the energy as a regularization term. This training framework is not only useful in neural architectures on hypersphere, but also effective in all types of standard neural networks such as multilayer perceptrons, convolutional neural networks, graph convolution networks, point cloud neural networks, etc.

Besides these aspects, many other aspects of neural networks can be revisited from a hyperspherical learning viewpoint, such as optimizers, normalization methods, etc. The hyperspherical learning framework provides a simple yet effective way to enable deep representation learning on hypersphere, and we believe it will benefit wider spectrum of deep learning applications.

CHAPTER 1
INTRODUCTION

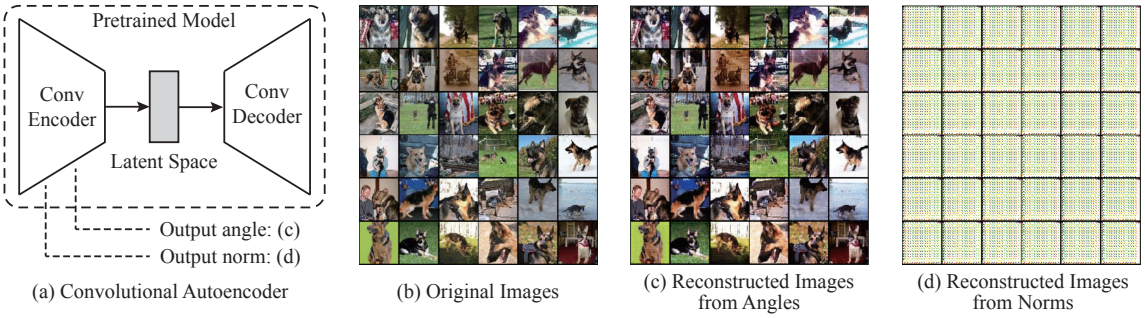
Recent years have witnessed tremendous progresses made by deep learning, such as visual recognition [1, 2], semantic segmentation [3] and object detection [4]. However, why highly over-parameterized deep neural networks can generalize well on unseen samples remains an open problem. Since neural networks have strong approximation power [5] and can approximate almost any non-linear function, the inductive bias of neural networks plays a crucial role in generalization. Therefore, how to effectively impose desirable inductive bias to neural networks becomes one of the core problems in deep learning, and it is still actively studied in different applications.

In this thesis, we present a novel deep representation learning framework that inherently introduces a hyperspherical inductive bias and shows superior empirical generalization in various supervised learning tasks. Specifically, learning deep representations on hypersphere requires us to revisit the components in typical neural networks, such as learning objectives (*i.e.*, loss functions), network architectures, and weight regularizations. We will revisit all these components and discuss how to modify them for adapting to the deep representation learning paradigm on hypersphere.

1.1 Motivation

Before delving deep into the details of the hyperspherical learning paradigm, we first discuss why learning on hypersphere is appealing and promising. We present several interesting observations that showcase the superiority of deep representation learning on hypersphere. These discoveries largely motivate our study.

Hyperspherical similarity is discriminative in high dimensions. As one of the most popular similarity measures in machine learning, Euclidean distance between two vectors



(a) Convolutional Autoencoder (b) Original Images (c) Reconstructed Images from Angles (d) Reconstructed Images from Norms

Figure 1.1: A toy convolutional autoencoder experiment to show that angular information can well preserve the semantics. Specifically, we first train a standard convolutional autoencoder, and fix the weights of the autoencoder during the experiment. We then make the convolutional encoder output either angles or norms instead of inner product during inference (with the weights of all the convolution kernels fixed).

$\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^n$ is defined as

$$d_E(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\| = \sqrt{\boldsymbol{x}^2 + \boldsymbol{y}^2 - 2\boldsymbol{x}^\top \boldsymbol{y}} \tag{1.1}$$

which will approach to $\sqrt{\boldsymbol{x}^2 + \boldsymbol{y}^2}$ when the dimension n becomes larger and larger. This is because random vectors (*i.e.*, each element of the vector follows an independent normal distribution) tend to be orthogonal to each other in high dimensions due to the law of large numbers. Therefore, Euclidean distance becomes less and less informative in high dimensions since it will be overwhelmed by the 2-norm of the two vectors.

As for the inner product $d_I(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\| \cdot \cos(\theta)$ which is also commonly used as a similarity measure in machine learning, it will encode some unnecessary information about the norm of the vectors. In terms of images, these norms are usually referring to the contrast and brightness, which are not essential for semantic visual recognition.

Therefore, the hyperspherical similarity which is a function of the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$ contains the most discriminative information to distinguish these vectors. A simple example of hyperspherical similarity is the cosine similarity $d_C = \cos(\theta)$ which is also equal to the normalized inner product $\frac{\boldsymbol{x}^\top \boldsymbol{y}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|}$.

Hyperspherical similarity preserves visual semantics. We conduct a toy experiment

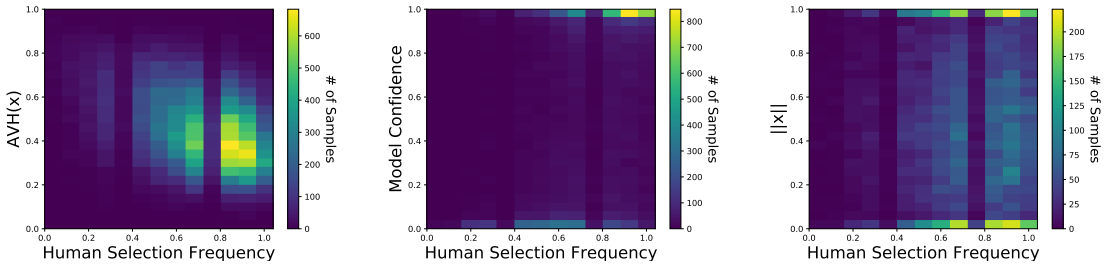


Figure 1.2: The left one presents Human Selection Frequency v.s. AVH, which we can see strong correlation. The second plot presents the correlation between HSF and Model Confidence with ResNet-50. The third one presents HSF v.s. $\|\mathbf{x}\|_2$. Note that different color indicates the density of samples in that bin.

to intuitively show that hyperspherical similarity can preserve most crucial visual semantic information. Specifically, we first train a convolutional auto-encoder on a set of images. After training, we fix the weights of the network and output either the angular information (*i.e.*, $\cos(\theta)$) or the magnitude information (*i.e.*, $\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|$) at the layer before the latent space, as illustrated in Figure 1.1(a). It is quite interesting to observe that the norm information alone does not give any meaningful output for the decoder, while angles can reconstruct images very well. Such an empirical observation suggests that hyperspherical similarity is able to largely preserve visual semantic information. In other words, most of the semantic difference in neural networks is encoded in hyperspherical similarity.

Hyperspherical similarity well correlates to human perception. To show that hyperspherical similarity is well aligned with human perception, we visualize the correlation between human selection frequency (HSF) [6] and model confidence, HSF and feature norm as well as HSF and angular visual hardness (AVH) [7]. HSF is defined as b/m if b out of m humans label a picture as the ground truth class, and it is a surrogate to the visual hardness of the image from human perspective. Model confidence is defined as $\frac{\exp(\boldsymbol{W}_y^\top \boldsymbol{x})}{\sum_i \exp(\boldsymbol{W}_i^\top \boldsymbol{x})}$ where $\boldsymbol{x}$ is the deep feature of the input sample with label y and $\boldsymbol{W}_i$ is the classifier of the i-th class in the final fully connected layer. AVH is defined as $\text{AVH} = \frac{\theta(\boldsymbol{W}_y, \boldsymbol{x})}{\sum_i \theta(\boldsymbol{W}_i, \boldsymbol{x})}$ where $\theta(\boldsymbol{W}_i, \boldsymbol{x})$ is the angle between the classifier of the i-th class and the input sample. It can

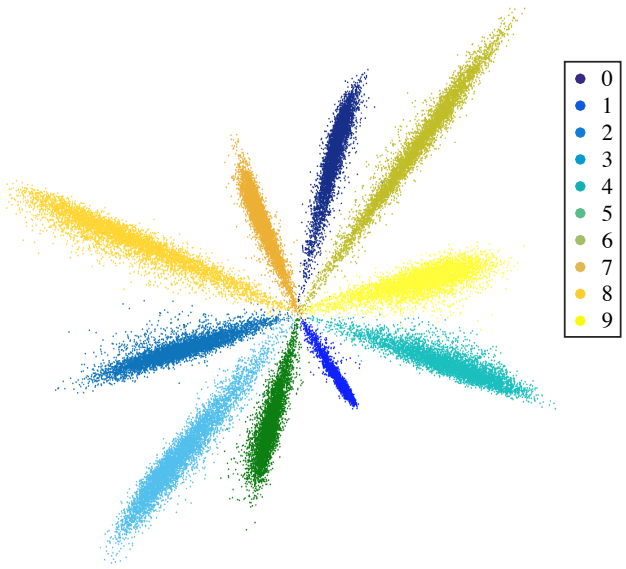


Figure 1.3: CNN learned features are naturally decoupled. These 2D features are output directly from the CNN by setting the feature dimension as 2.

be observed that AVH is based on the hyperspherical similarity between deep features and final classifiers from different classes. In general, smaller AVH indicates easier input image. From Figure 1.2, we can see that there is no obvious correlation between HSF and the norm. Model confidence shows that the network is over-confident about all the samples, since most samples lie on the top-right corner. Therefore, model confidence does not imply the intrinsic hardness that aligns with humans. In contrast, we can see that AVH shows very strong correlation with HSF, indicating that hyperspherical similarity is well aligned with human perception.

Deeply learned features already live on hypersphere. We visualize the deeply learned features of the MNIST training set [8] by setting the output dimension of the neural network as 2. Since we are not using any visualization tools, the visualized features will directly reflect the underlying feature distribution. From Figure 1.3, we can observe that these features that are directly output from the neural network are already quite discriminative on the hypersphere (*i.e.*, they can be well classified with angles). Therefore, deep neural networks are already implicitly learning features that are separable on a hypersphere. This observation motivates us to explicitly perform deep learning on hypersphere.

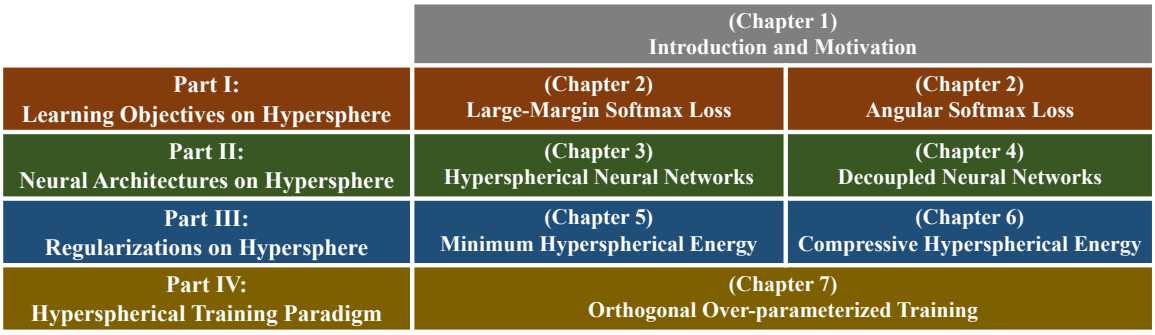


Figure 1.4: Overview of the thesis structure.

1.2 A Deep Representation Learning Framework on Hypersphere

We present a novel deep representation learning framework on hypersphere in this thesis. We introduce this framework from three aspects: learning objectives, neural architectures and regularizations. All of these are basic components of a standard neural network, and we will show how to adapt them to the hyperspherical learning framework in Part I, Part II and Part III. Last, we will introduce a new training paradigm for neural networks that is inspired by hyperspherical learning in Part IV. The structure of the thesis is shown in Figure 1.4.

1.2.1 Part I: Learning Objectives on Hypersphere

In order to enable learning deep features on hypersphere, the very first thing we should consider is the learning objectives (*i.e.*, loss functions). If we view the neural network as a black-box fitting machine that can approximate any complex function, the loss function will play the deciding role in guiding the neural network to learn the desirable deep features.

To explicitly constrain deep features to lie on a hypersphere, we consider several normalization methods that can be applied to the softmax cross-entropy loss. Based on these normalized loss functions, we propose a large-margin loss function that can learn features that not only lie on a hypersphere but also have large-margin separations among different classes. We also propose a few variants of this large-margin learning objective. Finally, we

apply it to both image classification and open-set face recognition, achieving the state-of-the-art performance.

1.2.2 Part II: Neural Architectures on Hypersphere

We further consider how to modify the standard neural architectures to benefit the learning on hypersphere, rather than viewing the neural network as a complete black box. Specifically, we revisit the convolution operator and propose a hyperspherical convolution operator that does not take norm into consideration when performing convolution. Furthermore, we generalize the hyperspherical operator to the decoupled convolution operator, encouraging stronger modeling flexibility. We show that these minor modifications on the convolution operator can ensure that the inference of the neural network does not depend on any activation norm.

Finally, we provide strong empirical evidence that shows superior generalization and adversarial robustness of both hyperspherical convolution and decoupled convolution in visual recognition.

1.2.3 Part III: Regularizations on Hypersphere

Because we do not consider norm in the hyperspherical learning framework, the widely used ℓ_2 weight decay is no longer useful. We need to come up with a new regularization that is suitable for learning on hypersphere. The new regularization should depend only on angles. A natural choice is the orthogonality regularization. In order to propose a more effective regularization, we draw inspiration from a well-known problem in physics – Thomson problem, where one seeks to find a state that distributes N electrons on a unit sphere as evenly as possible with minimum potential energy. In light of this intuition, we propose the minimum hyperspherical energy (MHE) as a generic regularization for neural networks. To further reduce the optimization difficulties of MHE, we propose the compressive minimum hyperspherical energy (CoMHE) and achieve better regularization

effect than MHE. Most importantly, we find that both MHE and CoMHE can be applied to standard convolutional neural networks (CNNs) and significantly improve generalization.

1.2.4 Part IV: Hyperspherical Training Paradigms

Following the intuition of the hyperspherical learning framework, we propose a novel training paradigm – orthogonal over-parameterized training (OPT), which can provably achieve MHE during training. We show that this training paradigm is generally useful and can applied to various types of neural networks (not limited to hyperspherical neural networks). In experiments, OPT can consistently improve the training stability and network generalization owing to the effective minimization of hyperspheriacl energy.

1.3 Thesis Contribution

This thesis makes conceptual and algorithmic contributions to the field of deep learning. Our contributions can be summarized as the following:

- We introduce a unified framework to perform deep learning on hypersphere. The framework is not only conceptually appealing but also very effective for improving generalization.
- We propose the large-margin learning objectives for the hyperspherical learning framework. The large-margin loss functions are able to learn deep features with excellent inter-class separability and intra-class compactness.
- We propose the hyperspherical networks and decoupled networks as the suitable neural architectures to perform deep learning on hypersphere. These neural architectures are shown to be useful for improving generalization and adversarial robustness.
- We propose the minimum hyperspherical energy regularization for the hyperspherical learning framework. MHE and its variants are very effective in preventing overfitting and improving generalization. More importantly, they are generally useful and

can be applied to various neural networks, not limited to the hyperspherical networks and decoupled networks.

- Inspired by the intuition of hyperspherical learning, we propose a general training paradigm for neural networks, called orthogonal over-parameterized training. OPT shows superior empirical performance in improving generalization of various neural networks.

1.4 Related Work

We survey the existing literature according to the three aspects (*i.e.*, learning objectives, neural architectures, and regularizations) of the hyperspherical learning framework.

1.4.1 Learning Objectives

Learning objective on hypersphere for CNNs has been an active topic since the pioneering large-margin softmax loss [9]. Inspired by this work, [10] has developed an angular softmax loss function which can produce desirable large angular margin, and shown impressive performance on face recognition. [11, 12] constrain the original softmax loss to work on the hypersphere by normalizing both classifier weights and features. In order to alleviate the training difficulty in [10], [13, 14, 15] have considered a new additive margin in the softmax loss, in contrast to the multiplicative margin used in [10]. There are also plenty follow-up work [16, 17, 18, 19] that further improves these large-margin losses. All these variants of the large-margin softmax loss on hypersphere are widely applied to visual recognition and face recognition, achieving the state-of-the-art performance.

1.4.2 Neural Architectures

In order to benefit the learning on hypersphere, the neural network architectures also need to be revisited. [12] first propose hyperspherical convolution operators that produces

angle instead of inner product. Based on this building block, [12] further introduces a hyperspherical neural network (SphereNet) that are distinct from conventional inner product based convolutional networks. Therefore, SphereNet is trained entirely based on the angles between kernels and local patches. Empirically, it shows faster convergence and stronger generalization compared to its CNN counterpart. [20] generalizes SphereNet to a more flexible neural network – decoupled networks (DCNets) by combining a magnitude function to hyperspherical convolutions. Quite interestingly, DCNets not only shows strong generalization power but also presents superior robustness against adversarial perturbations [21]. To address the few-shot classification problem, [22] propose the hyperspherical prototype networks that combines the idea of hyperspherical learning to the prototype network [23]. For generative modeling, hyperspherical learning can also be applied to develop new generative network architectures, such as sphere generative adversarial networks [24] and hyperspherical variational autoencoder [25].

1.4.3 Regularizations

Regularizations on hypersphere are more concerned with the angles among weights rather than the scale of weights (*i.e.*, weight decay). For example, SphereNet normalizes the weights of the convolution kernel during training, so weight decay can no longer serve as a regularization to control the network capacity. In order to introduce a suitable regularizaiton for SphereNet, [12] first considers to make the kernel weights to be orthogonal, which is called orthogonality regularization. In fact, there are plenty of work [26, 27, 28, 29, 30, 28] that focuses on characterizing diversity among neurons with orthogonality and regularizing the neural network by promoting the orthogonality.

Differently, [31] encourages the neurons to be uniformly distributed on the hypersphere to regularize the neural networks. Inspired by the Thomson problem in physics, MHE [31] and CoMHE [32] define the hyperspherical energy to characterize the diversity on a unit hypersphere and shows significant and consistent improvement in supervised learning tasks.

The importance of regularizing angular information is also discussed in [9, 12, 10, 20, 15, 13, 11, 14, 33, 22].

1.5 Thesis Organization

The following of the thesis will be organized in this way: In Part I (Chapter 2), we present the large-margin learning objectives on hypersphere. In Part II, we will discuss the neural architectures that can perform deep learning on hypersphere. Specifically, in Chapter 3, we will discuss how to design a network architecture (*i.e.*, SphereNet) that operates entirely on hypersphere. In Chapter 4, we generalize SphereNets to decoupled neural networks that have stronger generalization and adversarial robustness. Part III mainly discusses the alternative regularizations (other than the weight decay) in the hyperspherical learning framework. In Chapter 5, we will introduce a novel weight regularization for learning on hypersphere, called minimum hyperspherical energy. In Chapter 6, we propose the compressive minimum hyperspherical energy regularization to alleviate the optimization difficulty of MHE and improve the training stability. In Part IV, we will present a new training paradigm for neural networks (*i.e.*, orthogonal over-parameterized training) that is inspired by hyperspherical learning.

Part I:

Learning Objectives on Hypersphere

The learning objective on hypersphere plays an important role in our framework, since they directly determine the feature distribution and the final performance. There are many objective functions that can enable learning on hypersphere, such as triplet loss with normalization [34], contrastive loss with angular similarity [35], etc. Because we mostly focus on supervised learning tasks in the thesis, we build the learning objectives based on the popular cross-entropy loss with softmax. We denote the i-th input feature as $\boldsymbol{x}_i$ and its label as y_i. The original cross-entropy loss with softmax (we also call it softmax loss for convenience, and the full definition is given in section 2.1) can be written as

$$L_{\mathrm{o}} = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -\log\left(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\cos(\theta_{y_i})+b_{y_i}}}{\sum_j e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_j)+b_j}}\right)$$

where N is the number of samples, $\boldsymbol{W}_i$ denotes the linear classifier (in the final fully connected layer) for the i-the class and θ_j is the angle between the deep feature $\boldsymbol{x}_i$ and the classifier of the j-th class $\boldsymbol{W}_j$. In order to learn separable features on hypersphere, we can consider two simple normalization strategies for the softmax loss: weight normalization [12] and sphere normalization [12, 11].

Weight-normalized softmax loss. To make the classification fully dependent on the angle, we need to normalize the weights of the classifiers (*i.e.*, let $\|\boldsymbol{W}_j\| = 1, \forall j$) and remove all the bias terms:

$$L_{\mathrm{w}} = \frac{1}{N}\sum_i -\log\left(\frac{e^{\|\boldsymbol{x}_i\|\cos(\theta_{y_i})}}{\sum_j e^{\|\boldsymbol{x}_i\|\cos(\theta_j)}}\right)$$

where we can observe that as long as the angle between the feature and the target classifier is the smallest, then the classification is correct. In other word, the classification is purely based on the angles between the feature $\boldsymbol{x}_i$ and different classifiers $\boldsymbol{W}_j$.

Sphere-normalized softmax loss. Because the feature norm $\|\boldsymbol{x}_i\|$ is left in the softmax confidence in the weight-normalized softmax loss, the training dynamics for different input samples will be different. To remove the effects of the feature norm and make the decision

fully rely on the angles, we consider the following sphere-normalized softmax loss:

$$L_{\mathrm{s}} = \frac{1}{N}\sum_{i} -\log\left(\frac{e^{s_i\cdot\cos(\theta_{y_i})}}{\sum_j e^{s_i\cdot\cos(\theta_j)}}\right)$$

where s_i is the scale of the normalization and is typically used as a hyperparameter to alleviate the training difficulty. In order to make the classification purely dependent on the angles, we will use the same constant for all s_i, *i.e.*, $s_1 = s_2 = \cdots = s_C = s$ where C is the number of classes. In general, larger s leads to better training stability.

Both weight-normalized softmax loss and sphere-normalized softmax loss can enable the deep neural networks to learn separable features on hypersphere. Next, we will discuss the main topic of chapter 2 – large-margin learning objectives on hypersphere.

Generalized softmax loss. We generalize the original softmax loss by generalizing $\cos(\theta_{y_i})$ to a general function $\psi(\theta_{y_i})$ [9]:

$$L_{\mathrm{g}} = \frac{1}{N}\sum_{i} -\log\left(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\psi(\theta_{y_i})}}{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\psi(\theta_{y_i})} + \sum_{j\neq y_i} e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_j)}}\right)$$

based on which we present two different ways to impose the large angular margin constraint to the deep features. In order to learn deep features with large inter-class angular margin, we need to first make the confidence score $\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\psi(\theta_j)$ fully dependent on the angles. Both weight normalization and sphere normalization can be adopted here. To be more general, we consider the sphere normalization as an example, since the sphere-normalized softmax loss will reduce to the weight-normalized softmax loss when s_i becomes $\|\boldsymbol{x}_i\|$. Then we need to combine the large angular margin constraint to the features. To achieve this, we propose to make the classification confidence of the target class to be smaller than the standard classification confidence (which is based on $\cos(\theta_{y_i})$). Specifically, we only need to design a suitable $\psi(\theta_{y_i})$.

Multiplicative large-margin softmax loss. We propose to a multiplicative way [10] to

combine angular margin . The multiplicative large margin softmax loss is formulated as

$$L_{\mathrm{m}} = \frac{1}{N}\sum_{i} -\log\big(\frac{e^{s_i\psi(\theta_{y_i})}}{e^{s_i\psi(\theta_{y_i})}+\sum_{j\neq y_i} e^{s_i\cos(\theta_j)}}\big)$$

in which we define $\psi(\theta_{y_i}) = (-1)^k\cos(m\theta_{y_i}) - 2k$, $\theta_{y_i} \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]$ and $k \in [0, m-1]$. $m \geq 1$ is an integer that controls the size of the multiplicative angular margin. The intuition behind is that the multiplicative factor will make the classification more difficult than the original softmax loss, because the feature $\boldsymbol{x}_i$ needs to have smaller angle with the target classifier in order to be correctly classified. The multiplicative large-margin softmax loss (it is also called angular softmax loss in [10]) is one of the first loss functions for neural networks that can explicitly incorporate large margin on hypersphere. We will give the detailed introduction and derivation of this loss function in chapter 2.

Additive large-margin softmax loss. There are two additive ways [14, 13, 15] to impose the large angular margin. The first additive margin is to subtract a constant from $\cos(\theta_{y_i})$. Then we end up with the following additive large-margin softmax loss [14, 13]:

$$L_{\mathrm{a1}} = \frac{1}{N}\sum_{i} -\log\big(\frac{e^{s_i(\cos(\theta_{y_i})-m)}}{e^{s_i(\cos(\theta_{y_i})-m)}+\sum_{j\neq y_i} e^{s_i\cos(\theta_j)}}\big)$$

where m is a hyperparameter that controls scale of the additive angular margin. Larger m yields larger angular margin in general. Another way to impose the additive angular margin is to add a constant inside the cosine function [15]:

$$L_{\mathrm{a2}} = \frac{1}{N}\sum_{i} -\log\big(\frac{e^{s_i\cos(\theta_{y_i}+m)}}{e^{s_i\cos(\theta_{y_i}+m)}+\sum_{j\neq y_i} e^{s_i\cos(\theta_j)}}\big)$$

where m is a hyperparameter that controls scale of the additive angular margin. Larger m leads to larger angular margin in general.

One of the common characteristics of both multiplicative margin and additive margin is that the $\psi(\theta_{y_i})$ function has to be smaller than $\cos(\theta_{y_i})$ when $\theta_{y_i} \in [0, \pi]$. This design

criterion is the key to the large inter-class margin on hypersphere.

The next chapter is based on the following publications:

- *W. Liu*, Y. Wen, Z. Yu, M. Yang. Large-Margin Softmax Loss for Convolutional Neural Networks. *ICML 2016*
- *W. Liu*, Y. Wen, Z. Yu, M. Li, B. Raj, L. Song. SphereFace: Deep Hypersphere Embedding for Face Recognition. *CVPR 2017*

CHAPTER 2
LARGE-MARGIN LEARNING ON HYPERSPHERE

2.1 Preliminaries

Current widely used data loss functions in CNNs include Euclidean loss, (square) hinge loss, information gain loss, contrastive loss, triplet loss, Softmax loss, etc. In this thesis, we define the softmax loss as the combination of a cross-entropy loss, a softmax function and the last fully connected layer (see Figure 2.1). Under such definition, many prevailing CNN models can be viewed as the combination of a convolutional feature learning component and a softmax loss component, as shown in Figure 2.1. To enhance the intra-class compactness and inter-class separability, [36] trains the CNN with the combination of softmax loss and contrastive loss. The contrastive loss inputs the CNNs with pairs of training samples. If the input pair belongs to the same class, the contrastive loss will require their features are as similar as possible. Otherwise, the contrastive loss will require their distance larger than a margin. [34] uses the triplet loss to encourage a distance constraint similar to the contrastive loss. Differently, the triplet loss requires 3 (or a multiple of 3) training samples as input at a time. The triplet loss minimizes the distance between an anchor sample and a positive sample (of the same identity), and maximizes the distance between the anchor sample and a negative sample (of different identity). Both triplet loss and contrastive loss require a carefully designed pair selection procedure. Both [36] and [34] suggest that enforcing such a distance constraint that encourages intra-class compactness and inter-class separability can greatly boost the feature discriminativeness. Considering that the softmax loss is used in most supervised learning tasks, we aim to impose a margin constraint to the original softmax loss. Unlike any previous work, our work cast a novel view on generalizing the original softmax loss.

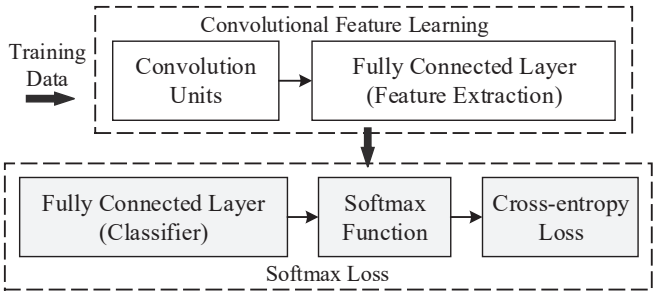


Figure 2.1: Standard CNNs can be viewed as convolutional feature learning machines that are supervised by the softmax loss.

We define the i-th input feature $\boldsymbol{x}_i$ with the label y_i. Then the original softmax loss can be written as

$$L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \tag{2.1}$$

where f_j denotes the j-th element ($j \in [1, K]$, K is the number of classes) of the vector of class scores $\boldsymbol{f}$, and N is the number of training data. In the softmax loss, $\boldsymbol{f}$ is usually the activations of a fully connected layer $\boldsymbol{W}$, so f_{y_i} can be written as $f_{y_i} = \boldsymbol{W}_{y_i}^T \boldsymbol{x}_i$ in which $\boldsymbol{W}_{y_i}$ is the y_i-th column of $\boldsymbol{W}$. Note that, we omit the constant b in $f_j, \forall j$ here to simplify discussion. Because f_j is the inner product between $\boldsymbol{W}_j$ and $\boldsymbol{x}_i$, it can be also formulated as $f_j = \|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_j)$ where θ_j ($0 \leq \theta_j \leq \pi$) is the angle between the vector $\boldsymbol{W}_j$ and $\boldsymbol{x}_i$. Thus the loss becomes

$$L_i = -\log\left(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\cos(\theta_{y_i})}}{\sum_j e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_j)}}\right) \tag{2.2}$$

2.2 Large-Margin Softmax Loss

2.2.1 Introduction

A recent trend towards learning with even stronger features is to reinforce CNNs with more discriminative information. Intuitively, the learned features are good if their intra-class compactness and inter-class separability are simultaneously maximized. While this may not be easy due to the inherent large intra-class variations in many tasks, the strong

representation ability of CNNs make it possible to learn invariant features towards this direction. Inspired by such idea, the contrastive loss [37] and triplet loss [34] were proposed to enforce extra intra-class compactness and inter-class separability. A consequent problem, however, is that the number of training pairs and triplets can theoretically go up to $\mathcal{O}(N^2)$ where N is the total number of training samples. Considering that CNNs often handle large-scale training sets, a subset of training samples need to be carefully selected for these losses. The softmax function is widely adopted by many CNNs [1, 2, 38] due to its simplicity and probabilistic interpretation. Together with the cross-entropy loss, they form arguably one of the most commonly used components in CNN architectures. Despite its popularity, current softmax loss does not explicitly encourage intra-class compactness and inter-class-separability. Our key intuition is that the separability between sample and parameter can be factorized into amplitude ones and angular ones with cosine similarity: $\boldsymbol{W}_c\boldsymbol{x} = \|\boldsymbol{W}_c\|_2\|\boldsymbol{x}\|_2\cos(\theta_c)$, where c is the class index, and the corresponding parameters $\boldsymbol{W}_c$ of the last fully connected layer can be regarded as the linear classifier of class c. Under softmax loss, the label prediction decision rule is largely determined by the angular similarity to each class since softmax loss uses cosine distance as classification score. The purpose of this work, therefore, is to generalize the softmax loss to a more general large-margin softmax (L-Softmax) loss in terms of angular similarity, leading to potentially larger angular separability between learned features. This is done by incorporating a preset constant m multiplying with the angle between sample and the classifier of ground truth class. m determines the strength of getting closer to the ground truth class, producing an angular margin. One shall see, the conventional softmax loss becomes a special case of the L-Softmax loss under our proposed framework. Our idea is verified by Figure 2.2 where the learned features by L-Softmax become much more compact and well separated.

The L-Softmax loss is a flexible learning objective with adjustable inter-class angular margin constraint. It presents a learning task of adjustable difficulty where the difficulty gradually increases as the required margin becomes larger. The L-Softmax loss has sev-

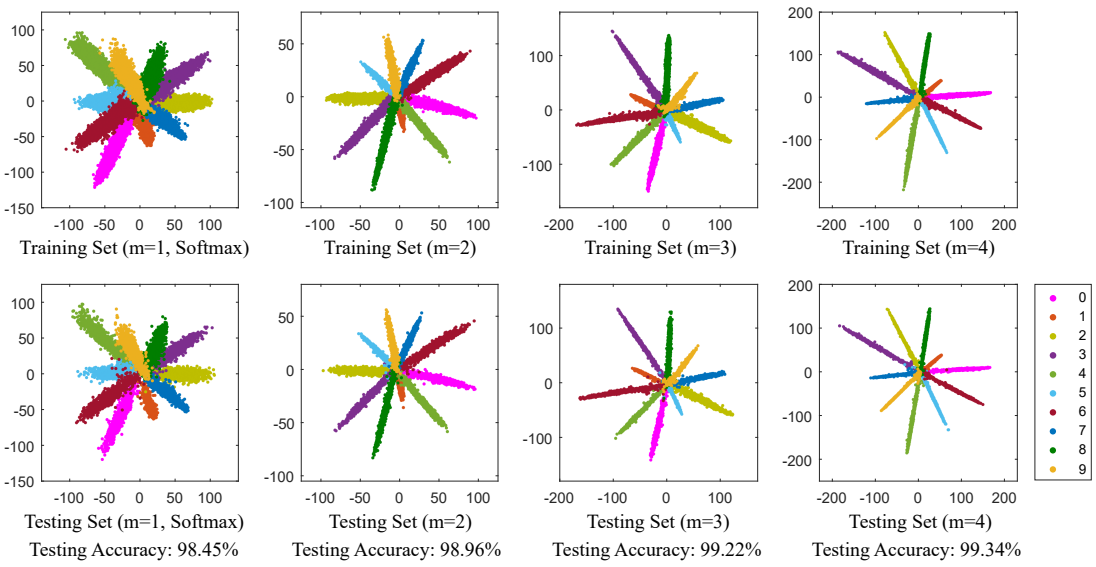


Figure 2.2: CNN-leanrned features visualization (Softmax Loss (m=1) vs. L-Softmax loss (m=2,3,4)) in MNIST dataset. Specifically, we set the feature (input of the L-Softmax loss) dimension as 2, and then plot them by class.

eral desirable advantages. First, it encourages angular decision margin between classes, generating more discriminative features. Its geometric interpretation is very clear and intuitive, as elaborated in Section 3.2. Second, it partially avoids overfitting by defining a more difficult learning target, casting a different viewpoint to the overfitting problem. Third, L-Softmax benefits not only classification problems, but also verification problems where ideally learned features should have the minimum inter-class distance being greater than the maximum intra-class distance. In this case, learning well separated features can significantly improve the performance.

2.2.2 Intuition

We give a simple example to describe our intuition. Consider the binary classification and we have a sample $\boldsymbol{x}$ from class 1. The original softmax is to force $\boldsymbol{W}_1^T\boldsymbol{x} > \boldsymbol{W}_2^T\boldsymbol{x}$ (i.e. $\|\boldsymbol{W}_1\|\|\boldsymbol{x}\|\cos(\theta_1) > \|\boldsymbol{W}_2\|\|\boldsymbol{x}\|\cos(\theta_2)$) in order to classify $\boldsymbol{x}$ correctly. However, we want to make the classification more rigorous in order to produce a decision margin. So we instead require $\|\boldsymbol{W}_1\|\|\boldsymbol{x}\|\cos(m\theta_1) > \|\boldsymbol{W}_2\|\|\boldsymbol{x}\|\cos(\theta_2)$ ($0 \leq \theta_1 \leq \frac{\pi}{m}$) where m is a

positive integer. Because the following inequality holds:

$$\begin{aligned} \|\boldsymbol{W}_1\|\|\boldsymbol{x}\|\cos(\theta_1) &\geq \|\boldsymbol{W}_1\|\|\boldsymbol{x}\|\cos(m\theta_1) \\ &> \|\boldsymbol{W}_2\|\|\boldsymbol{x}\|\cos(\theta_2). \end{aligned} \tag{2.3}$$

Therefore, $\|\boldsymbol{W}_1\|\|\boldsymbol{x}\|\cos(\theta_1) > \|\boldsymbol{W}_2\|\|\boldsymbol{x}\|\cos(\theta_2)$ has to hold. So the new classification criteria is a stronger requirement to correctly classify $\boldsymbol{x}$, producing a more rigorous decision boundary for class 1.

2.2.3 Definition

Following the notation in the preliminaries, the L-Softmax loss is defined as

$$L_i = -\log\left(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\psi(\theta_{y_i})}}{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\psi(\theta_{y_i})} + \sum_{j\neq y_i} e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_j)}}\right) \tag{2.4}$$

in which we generally require

$$\psi(\theta) = \begin{cases} \cos(m\theta), & 0 \leq \theta \leq \dfrac{\pi}{m} \\ \mathcal{D}(\theta), & \dfrac{\pi}{m} < \theta \leq \pi \end{cases} \tag{2.5}$$

where m is a integer that is closely related to the classification margin. With larger m, the classification margin becomes larger and the learning objective also becomes harder. Meanwhile, $\mathcal{D}(\theta)$ is required to be a monotonically decreasing function and $\mathcal{D}(\frac{\pi}{m})$ should equal $\cos(\frac{\pi}{m})$.

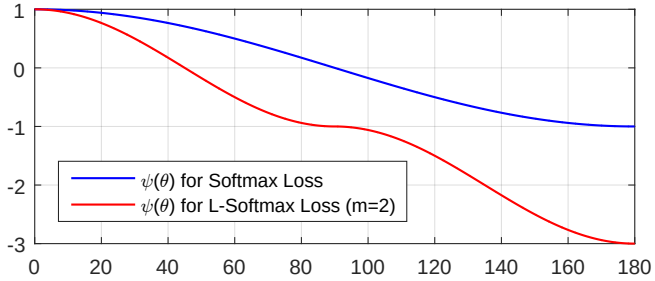


Figure 2.3: $\psi(\theta)$ for softmax loss and L-Softmax loss.

To simplify the forward and backward propagation, we construct a specific $\psi(\theta_i)$ here:

$$\psi(\theta) = (-1)^k \cos(m\theta) - 2k, \quad \theta \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}] \tag{2.6}$$

where $k \in [0, m-1]$ and k is an integer. Combining Equation 2.1, Equation 2.4 and Equation 2.6, we have the final L-Softmax loss. For forward and backward propagation, we need to replace $\cos(\theta_j)$ with $\frac{\boldsymbol{W}_j^T \boldsymbol{x}_i}{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|}$, and replace $\cos(m\theta_{y_i})$ with

$$\begin{aligned} \cos(m\theta_{y_i}) = {} & C_m^0 \cos^m(\theta_{y_i}) - C_m^2 \cos^{m-2}(\theta_{y_i})(1 - \cos^2(\theta_{y_i})) \\ & + C_m^4 \cos^{m-4}(\theta_{y_i})(1 - \cos^2(\theta_{y_i}))^2 + \cdots \\ & (-1)^n C_m^{2n} \cos^{m-2n}(\theta_{y_i})(1 - \cos^2(\theta_{y_i}))^n + \cdots \end{aligned} \tag{2.7}$$

where n is an integer and $2n \leq m$. After getting rid of θ, we could perform derivation with respect to $\boldsymbol{x}$ and $\boldsymbol{W}$. It is also trivial to perform derivation with mini-batch input.

2.2.4 Geometric Interpretation

We aim to encourage an angle margin between classes via the L-Softmax loss. To simplify the geometric interpretation, we analyze the binary classification case where there are only $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$.

First, we consider the $\|\boldsymbol{W}_1\| = \|\boldsymbol{W}_2\|$ scenario as shown in Figure 2.4. With $\|\boldsymbol{W}_1\| = \|\boldsymbol{W}_2\|$, the classification result depends entirely on the angles between $\boldsymbol{x}$ and $\boldsymbol{W}_1(\boldsymbol{W}_2)$. In the training stage, the original softmax loss requires $\theta_1 < \theta_2$ to classify the sample $\boldsymbol{x}$ as class 1, while the L-Softmax loss requires $m\theta_1 < \theta_2$ to make the same decision. We can see the L-Softmax loss is more rigor about the classification criteria, which leads to a classification margin between class 1 and class 2. If we assume both softmax loss and L-Softmax loss are optimized to the same value and all training features can be perfectly classified, then the angle margin between class 1 and class 2 is given by $\frac{m-1}{m+1}\theta_{1,2}$ where $\theta_{1,2}$ is the angle between classifier vector $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. The L-Softmax loss also makes

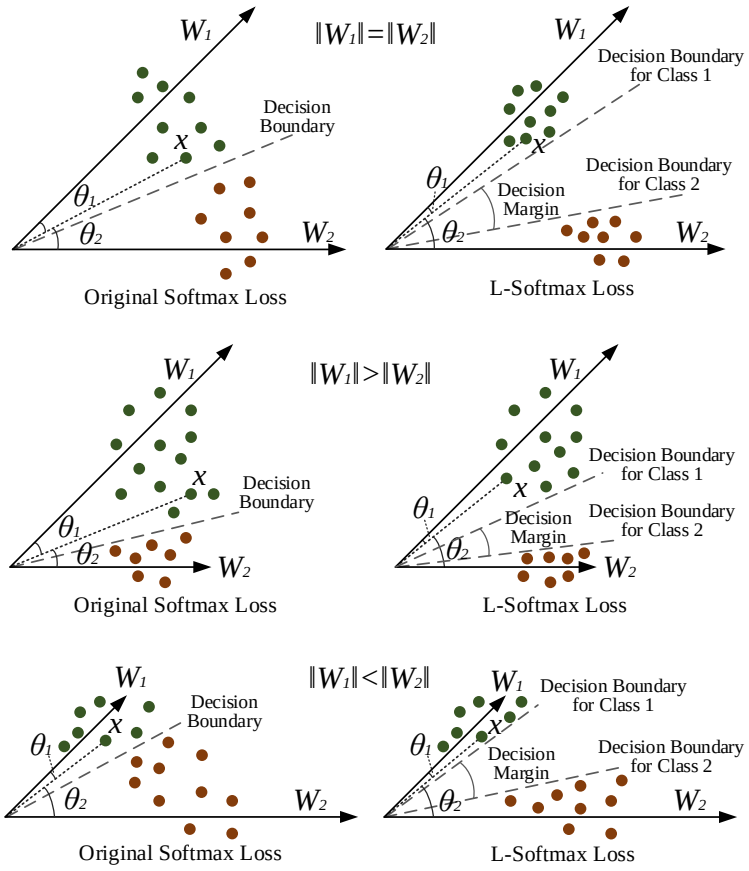


Figure 2.4: Examples of Geometric Interpretation.

the decision boundaries for class 1 and class 2 different as shown in Figure 2.4, while originally the decision boundaries are the same. From another viewpoint, we let $\theta_1' = m\theta_1$ and assume that both the original softmax loss and the L-Softmax loss can be optimized to the same value. Then we can know θ_1' in the original softmax loss is $m-1$ times larger than θ_1 in the L-Softmax loss. As a result, the angle between the learned feature and $\boldsymbol{W}_1$ will become smaller. For every class, the same conclusion holds. In essence, the L-Softmax loss narrows the feasible angle[1] for every class and produces an angle margin between these classes.

For both the $\|\boldsymbol{W}_1\| > \|\boldsymbol{W}_2\|$ and $\|\boldsymbol{W}_1\| < \|\boldsymbol{W}_2\|$ scenarios, the geometric interpretation is a bit more complicated. Because the length of $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ is different, the feasible

[1]Feasible angle of the i-th class refers to the possible angle between $\boldsymbol{x}$ and $\boldsymbol{W}_i$ that is learned by CNNs.

angles of class 1 and class 2 are also different (see the decision boundary of original softmax loss in Figure 2.4). Normally, the larger $\boldsymbol{W}_j$ is, the larger the feasible angle of its corresponding class is. As a result, the L-Softmax loss also produces different feasible angles for different classes. Similar to the analysis of the $\|\boldsymbol{W}_1\| = \|\boldsymbol{W}_2\|$ scenario, the proposed loss will also generate a decision margin between class 1 and class 2.

2.2.5 Discussion

The L-Softmax loss utilizes a simple modification over the original softmax loss, achieving a classification angle margin between classes. By assigning different values for m, we define a flexible learning task with adjustable difficulty for CNNs. The L-Softmax loss is endowed with some nice properties such as

- The L-Softmax loss has a clear geometric interpretation. m controls the margin among classes. With bigger m (under the same training loss), the ideal margin between classes becomes larger and the learning difficulty is also increased. With $m = 1$, the L-Softmax loss becomes identical to the original softmax loss.

- The L-Softmax loss defines a relatively difficult learning objective with adjustable margin (difficulty). A difficult learning objective can effectively avoid over-fitting and take full advantage of the strong learning ability from deep and wide architectures.

- The L-Softmax loss can be easily used as a drop-in replacement for standard loss, as well as used in tandem with other performance-boosting approaches and modules, including learning activation functions, data augmentation, pooling functions or other modified network architectures.

2.2.6 Optimization

It is easy to compute the forward and backward propagation for the L-Softmax loss, so it is also trivial to optimize the L-Softmax loss using typical stochastic gradient descent. For L_i, the only difference between the original softmax loss and the L-Softmax loss lies in f_{y_i}. Thus we only need to compute f_{y_i} in forward and backward propagation while $f_j, j \neq y_i$ is the same as the original softmax loss. Putting in Equation 2.6 and Equation 2.7, f_{y_i} is written as

$$\begin{aligned} f_{y_i} =&(-1)^k \cdot \|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\| \cos(m\theta_i) - 2k \cdot \|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\| \\ =&(-1)^k \cdot \|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\| \bigg(C_m^0 \big(\frac{\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|}\big)^m - \\ & C_m^2 \big(\frac{\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|}\big)^{m-2} \big(1 - \big(\frac{\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|}\big)^2\big) + \cdots \bigg) - 2k \cdot \|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\| \end{aligned} \tag{2.8}$$

where $\frac{\boldsymbol{W}_{y_i}^T \boldsymbol{x}}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}\|} \in [\cos(\frac{k\pi}{m}), \cos(\frac{(k+1)\pi}{m})]$ and k is an integer that belongs to $[0, m-1]$. For the backward propagation, we use the chain rule to compute the partial derivative: $\frac{\partial L_i}{\partial \boldsymbol{x}_i} = \sum_j \frac{\partial L_i}{\partial f_j} \frac{\partial f_j}{\partial \boldsymbol{x}_i}$ and $\frac{\partial L_i}{\partial \boldsymbol{W}_{y_i}} = \sum_j \frac{\partial L_i}{\partial f_j} \frac{\partial f_j}{\partial \boldsymbol{W}_{y_i}}$. Because $\frac{\partial L_i}{\partial f_j}$ and $\frac{\partial f_j}{\partial \boldsymbol{x}_i}, \frac{\partial f_j}{\partial \boldsymbol{W}_{y_i}}, \forall j \neq y_i$ are the same for both original softmax loss and L-Softmax loss, we leave it out for simplicity. $\frac{\partial f_{y_i}}{\partial \boldsymbol{x}_i}$ and $\frac{\partial f_{y_i}}{\partial \boldsymbol{W}_{y_i}}$ can be computed via

$$\begin{aligned} \frac{\partial f_{y_i}}{\partial \boldsymbol{x}_i} = (-1)^k \cdot \bigg(& C_m^0 \frac{m(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^{m-1} \boldsymbol{W}_{y_i}}{(\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|)^{m-1}} - C_m^0 \frac{(m-1)(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^m \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|^{m-1}\|\boldsymbol{x}_i\|^{m+1}} \\ & - C_m^2 \frac{(m-2)(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^{m-3} \boldsymbol{W}_{y_i}}{(\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|)^{m-3}} + C_m^2 \frac{(m-3)(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^{m-2} \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|^{m-3}\|\boldsymbol{x}_i\|^{m-1}} \\ & + C_m^2 \frac{m(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^{m-1} \boldsymbol{W}_{y_i}}{(\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|)^{m-1}} - C_m^2 \frac{(m-1)(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^m \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|^{m-1}\|\boldsymbol{x}_i\|^{m+1}} + \cdots \bigg) - 2k \cdot \frac{\|\boldsymbol{W}_{y_i}\| \boldsymbol{x}_i}{\|\boldsymbol{x}_i\|}, \end{aligned} \tag{2.9}$$

$$\frac{\partial f_{y_i}}{\partial \boldsymbol{W}_{y_i}} = (-1)^k \cdot \left(C_m^0 \frac{m(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^{m-1} \boldsymbol{x}_i}{(\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|)^{m-1}} - C_m^0 \frac{(m-1)(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^m \boldsymbol{W}_{y_i}}{\|\boldsymbol{W}_{y_i}\|^{m+1}\|\boldsymbol{x}_i\|^{m-1}} \right.$$
$$- C_m^2 \frac{(m-2)(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^{m-3} \boldsymbol{x}_i}{(\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|)^{m-3}} + C_m^2 \frac{(m-3)(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^{m-2} \boldsymbol{W}_{y_i}}{\|\boldsymbol{W}_{y_i}\|^{m-1}\|\boldsymbol{x}_i\|^{m-3}}$$
$$\left. + C_m^2 \frac{m(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^{m-1} \boldsymbol{x}_i}{(\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|)^{m-1}} - C_m^2 \frac{(m-1)(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^m \boldsymbol{W}_{y_i}}{\|\boldsymbol{W}_{y_i}\|^{m+1}\|\boldsymbol{x}_i\|^{m-1}} + \cdots \right) - 2k \cdot \frac{\|\boldsymbol{x}_i\| \boldsymbol{W}_{y_i}}{\|\boldsymbol{W}_{y_i}\|}. \tag{2.10}$$

In implementation, k can be efficiently computed by constructing a look-up table for $\frac{\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|}$ (i.e. $\cos(\theta_{y_i})$). To be specific, we give an example of the forward and backward propagation when $m = 2$. Thus f_i is written as

$$f_i = (-1)^k \frac{2(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^2}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|} - \big(2k + (-1)^k\big)\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\| \tag{2.11}$$

where, $k = \begin{cases} 1, & \frac{\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|} \leq \cos(\frac{\pi}{2}) \\ 0, & \frac{\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|} > \cos(\frac{\pi}{2}) \end{cases}$.

In the backward propagation, $\frac{\partial f_{y_i}}{\partial \boldsymbol{x}_i}$ and $\frac{\partial f_{y_i}}{\partial \boldsymbol{W}_{y_i}}$ can be computed with

$$\frac{\partial f_{y_i}}{\partial \boldsymbol{x}_i} = (-1)^k \left(\frac{4\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i \boldsymbol{W}_{y_i}}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|} - \frac{2(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^2 \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|^3} \right) - \big(2k + (-1)^k\big) \frac{\|\boldsymbol{W}_{y_i}\| \boldsymbol{x}_i}{\|\boldsymbol{x}_i\|}, \tag{2.12}$$

$$\frac{\partial f_{y_i}}{\partial \boldsymbol{W}_{y_i}} = (-1)^k \left(\frac{4\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i \boldsymbol{x}_i}{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|} - \frac{2(\boldsymbol{W}_{y_i}^T \boldsymbol{x}_i)^2 \boldsymbol{W}_{y_i}}{\|\boldsymbol{x}_i\|\|\boldsymbol{W}_{y_i}\|^3} \right) - \big(2k + (-1)^k\big) \frac{\|\boldsymbol{x}_i\| \boldsymbol{W}_{y_i}}{\|\boldsymbol{W}_{y_i}\|}. \tag{2.13}$$

While $m \geq 3$, we can still use Equation 2.8, Equation 2.9 and Equation 2.10 to compute the forward and backward propagation.

Layer	MNIST (for Figure 2.2)	MNIST	CIFAR10/CIFAR10+	CIFAR100	LFW
Conv0.x	N/A	[3×3, 64]×1	[3×3, 64]×1	[3×3, 96]×1	[3×3, 64]×1, Stride 2
Conv1.x	[5×5, 32]×2, Padding 2	[3×3, 64]×3	[3×3, 64]×4	[3×3, 96]×4	[3×3, 64]×4
Pool1	2×2 Max, Stride 2				
Conv2.x	[5×5, 64]×2, Padding 2	[3×3, 64]×3	[3×3, 96]×4	[3×3, 192]×4	[3×3, 256]×4
Pool2	2×2 Max, Stride 2				
Conv3.x	[5×5, 128]×2, Padding 2	[3×3, 64]×3	[3×3, 128]×4	[3×3, 384]×4	[3×3, 256]×4
Pool3	2×2 Max, Stride 2				
Conv4.x	N/A	N/A	N/A	N/A	[3×3, 256]×4
FC	2	256	256	512	512

Table 2.1: Our CNN architectures for different benchmark datasets. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers. E.g., [3×3, 64]×4 denotes 4 cascaded convolution layers with 64 filters of size 3×3.

2.2.7 Experiments and Results

Experimental Settings

We evaluate the generalized softmax loss in two typical vision applications: visual classification and face verification. In visual classification, we use three standard benchmark datasets: MNIST [39], CIFAR10 [40], and CIFAR100 [40]. In face verification, we evaluate our method on the widely used LFW dataset [41]. We only use a single model in all baseline CNNs to compare our performance. For convenience, we use L-Softmax to denote the L-Softmax loss. Both Softmax and L-Softmax in the experiments use the same CNN shown in Table 2.1.

General Settings: We follow the design philosophy of VGG-net [42] in two aspects: (1) for convolution layers, the kernel size is 3×3 and 1 padding (if not specified) to keep the feature map unchanged. (2) for pooling layers, if the feature map size is halved, the number of filters is doubled in order to preserve the time complexity per layer. Our CNN architectures are described in Table 2.1. In convolution layers, the stride is set to 1 if not specified. We implement the CNNs using the Caffe library [43] with our modifications. For all experiments, we adopt the PReLU [38] as the activation functions, and the batch size is 256. We use a weight decay of 0.0005 and momentum of 0.9. The weight initialization in [38] and batch normalization [44] are used in our networks but without dropout. Note that we only perform the mean substraction preprocessing for training and testing data. For optimiza-

Table 2.2: Recognition error rate (%) on MNIST dataset.

Method	Error Rate
CNN [45]	0.53
DropConnect [46]	0.57
FitNet [47]	0.51
NiN [48]	0.47
Maxout [49]	0.45
DSN [50]	0.39
R-CNN [51]	**0.31**
GenPool [52]	**0.31**
Hinge Loss	0.47
Softmax	0.40
L-Softmax (m=2)	0.32
L-Softmax (m=3)	**0.31**
L-Softmax (m=4)	**0.31**

tion, normally the stochastic gradient descent will work well. However, when training data has too many subjects (such as CASIA-WebFace dataset), the convergence of L-Softmax will be more difficult than softmax loss. For those cases that L-Softmax has difficulty converging, we use a learning strategy by letting $f_{y_i} = \frac{\lambda\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\cos(\theta_{y_i})+\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\psi(\theta_{y_i})}{1+\lambda}$ and start the gradient descent with a very large λ (it is similar to optimize the original softmax). Then we gradually reduce λ during iteration. Ideally λ can be gradually reduced to zero, but in practice, a small value will usually suffice.

MNIST, CIFAR10, CIFAR100: We start with a learning rate of 0.1, divide it by 10 at 12k and 15k iterations, and eventually terminate training at 18k iterations, which is determined on a 45k/5k train/val split.

Face Verification: The learning rate is set to 0.1, 0.01, 0.001 and is switched when the training loss plateaus. The total number of epochs is about is about 30 for our models.

Testing: we use the softmax to classify the testing samples in MNIST, CIFAR10 and CIFAR100 dataset. In LFW dataset, we use the simple cosine distance and the nearest neighbor rule for face verification.

Table 2.3: Recognition error rate (%) on CIFAR10 dataset. CIFAR10 denotes the performance without data augmentation, while CIFAR10+ is with data augmentation.

Method	CIFAR10	CIFAR10+
DropConnect [46]	9.41	9.32
FitNet [47]	N/A	8.39
NiN + LA units [48]	10.47	8.81
Maxout [49]	11.68	9.38
DSN [50]	9.69	7.97
All-CNN [53]	9.08	7.25
R-CNN [51]	8.69	7.09
ResNet [2]	N/A	6.43
GenPool [52]	7.62	6.05
Hinge Loss	9.91	6.96
Softmax	9.05	6.50
L-Softmax (m=2)	7.73	6.01
L-Softmax (m=3)	7.66	5.94
L-Softmax (m=4)	**7.58**	**5.92**

Visual Classification

MNIST: Our network architecture is shown in Table 2.1. Table 2.2 shows the previous best results and those for our proposed L-Softmax loss. From the results, the L-Softmax loss not only outperforms the original softmax loss using the same network but also achieves the state-of-the-art performance compared to the other deep CNN architectures. In Figure 2.2, we also visualize the learned features by the L-Softmax loss and compare them to the original softmax loss. Figure 2.2 validates the effectiveness of the large margin constraint within L-Softmax loss. With larger m, we indeed obtain a larger angular decision margin.

CIFAR10: We use two commonly used comparison protocols in CIFAR10 dataset. We first compare our L-Softmax loss under no data augmentation setup. For the data augmentation experiment, we follow the standard data augmentation in [50] for training: 4 pixels are padded on each side, and a 32×32 crop is randomly sampled from the padded image or its horizontal flip. In testing, we only evaluate the single view of the original 32×32 image. The results are shown in Table 2.3. One can observe that our L-Softmax loss greatly boosts the accuracy, achieving 1%-2% improvement over the original softmax loss and the other

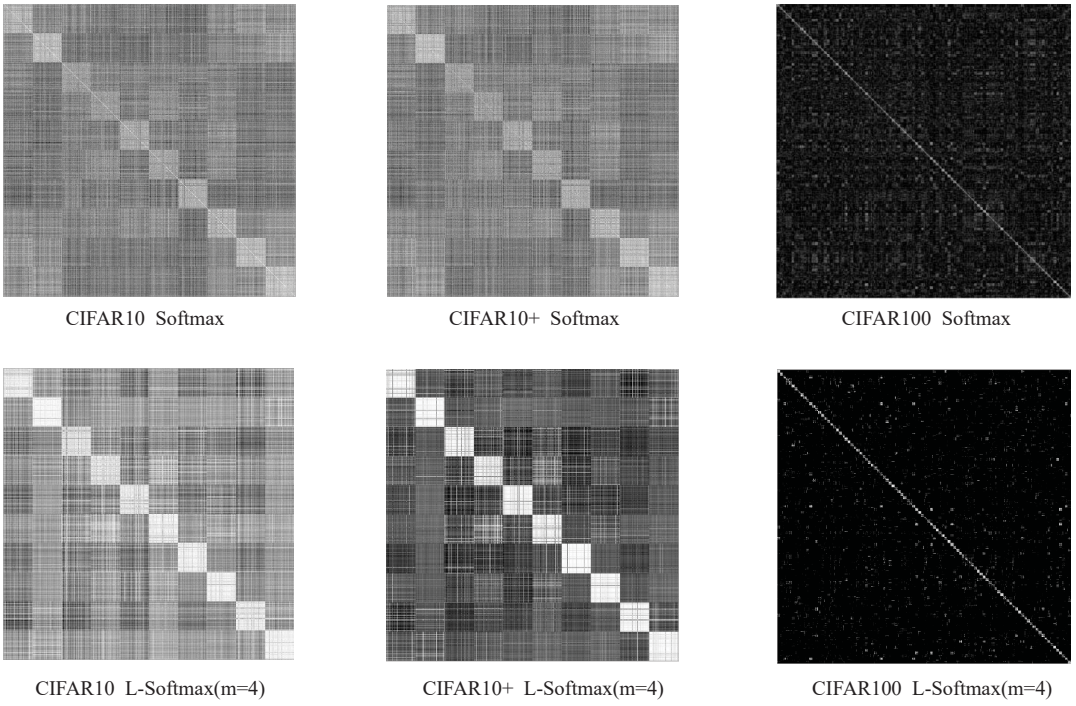


Figure 2.5: Confusion matrix on CIFAR10, CIFAR10+ and CIFAR100.

state-of-the-art CNNs.

CIFAR100: We also evaluate the generalize softmax loss on the CIFAR100 dataset. The CNN architecture refers to Table 2.1. One can notice that the L-Softmax loss outperform the CNN with softmax loss and all the other competitive methods. From Table 2.4, the L-Softmax loss improves more than 2.5% accuracy over the CNN and more than 1% over the current state-of-the-art CNN.

Confusion Matrix Visualization: We also give the confusion matrix comparison between the softmax baseline and the L-Softmax loss (m=4) in Figure 2.5. Specifically we normalize the learned features and then calculate the cosine distance between these features. From Figure 2.5, one can see that the intra-class compactness is greatly enhanced while the inter-class separability is also enlarged.

Table 2.4: Recognition error rate (%) on CIFAR100 dataset.

Method	Error Rate
FitNet [47]	35.04
NiN [48]	35.68
Maxout [49]	38.57
DSN [50]	34.57
dasNet [54]	33.78
All-CNN [53]	33.71
R-CNN [51]	31.75
GenPool [52]	32.37
Hinge Loss	32.90
Softmax	32.74
L-Softmax (m=2)	29.95
L-Softmax (m=3)	29.87
L-Softmax (m=4)	**29.53**

Table 2.5: Verification performance (%) on LFW dataset. * denotes the outside data is private (not publicly available).

Method	Outside Data	Accuracy
FaceNet [34]	200M*	**99.65**
Deep FR [55]	2.6M	98.95
DeepID2+ [56]	300K*	98.70
Yi et al. [57]	WebFace	97.73
Ding et al. [58]	WebFace	98.43
Softmax	WebFace	96.53
Softmax + Contrastive	WebFace	97.31
L-Softmax (m=2)	WebFace	97.81
L-Softmax (m=3)	WebFace	98.27
L-Softmax (m=4)	WebFace	**98.71**

Face Verification

To further evaluate the learned features, we conduct an experiment on the famous LFW dataset [41]. The dataset collects 13,233 face images from 5749 persons from uncontrolled conditions. Following the unrestricted with labeled outside data protocol [41], we train on the publicly available CASIA-WebFace [57] outside dataset (490k labeled face images belonging to over 10,000 individuals) and test on the 6,000 face pairs on LFW. People overlapping between the outside training data and the LFW testing data are excluded. As preprocessing, we use IntraFace [59] to align the face images and then crop them based on 5 points. Then we train a single network for feature extraction, so we only compare

the single model performance of current state-of-the-art CNNs. Finally PCA is used to form a compact feature vector. The results are given in Table 2.5. The generalize softmax loss achieves the current best results while only trained with the CASIA-WebFace outside data, and is also comparable to the current state-of-the-art CNNs with private outside data. Experimental results well validate the conclusion that the L-Softmax loss encourages the intra-class compactness and inter-class separability.

2.3 Angular Softmax Loss

2.3.1 Introduction

Recent years have witnessed the great success of convolutional neural networks (CNNs) in face recognition (FR). Owing to advanced network architectures [1, 42, 60, 61] and discriminative learning approaches [62, 34, 63], deep CNNs have boosted the FR performance to an unprecedent level. Typically, face recognition can be categorized as face identification and face verification [64, 65]. The former classifies a face to a specific identity, while the latter determines whether a pair of faces belongs to the same identity.

In terms of testing protocol, face recognition can be evaluated under closed-set or open-set settings, as illustrated in Figure 2.6. For closed-set protocol, all testing identities are predefined in training set. It is natural to classify testing face images to the given identities. In this scenario, face verification is equivalent to performing identification for a pair of faces respectively (see left side of Figure 2.6). Therefore, closed-set FR can be well addressed as a classification problem, where features are expected to be separable. For open-set protocol, the testing identities are usually disjoint from the training set, which makes FR more challenging yet close to practice. Since it is impossible to classify faces to known identities in training set, we need to map faces to a discriminative feature space. In this scenario, face identification can be viewed as performing face verification between the probe face and every identity in the gallery (see right side of Figure 2.6). Open-set FR is essentially a metric learning problem, where the key is to learn discriminative large-margin

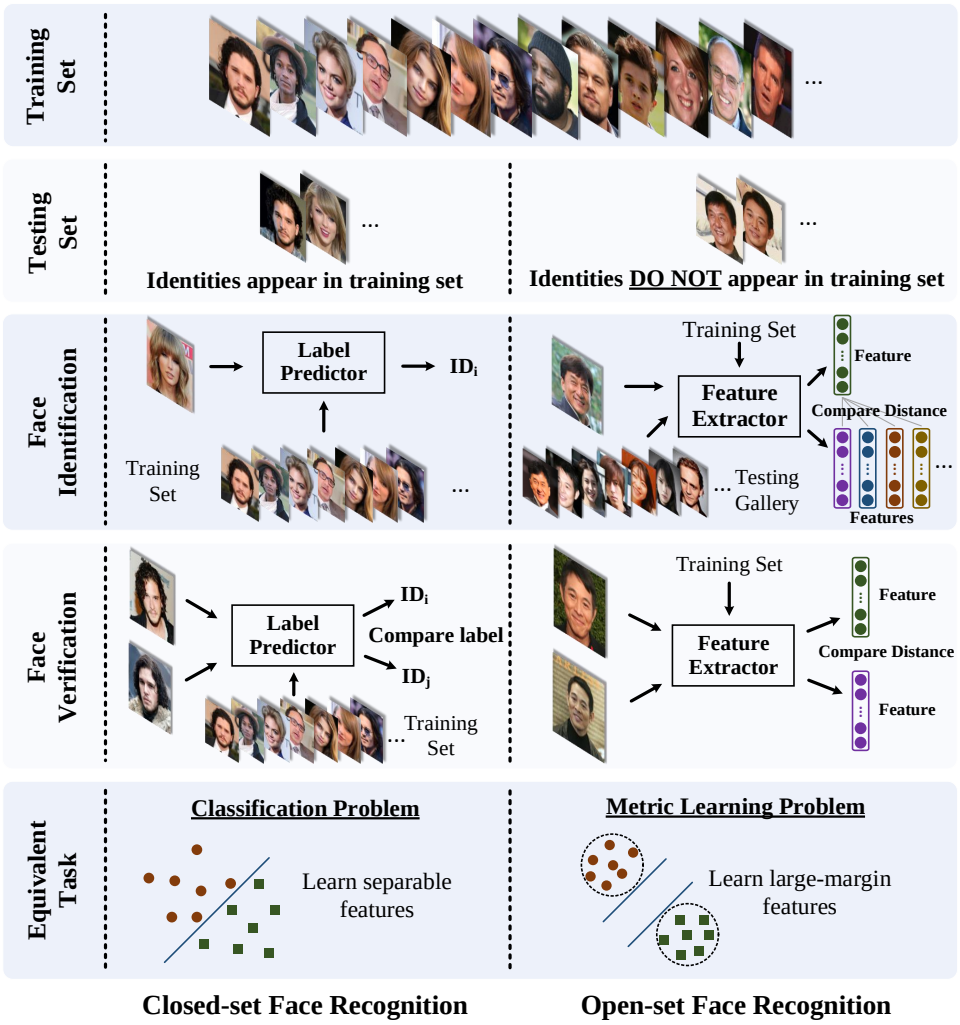
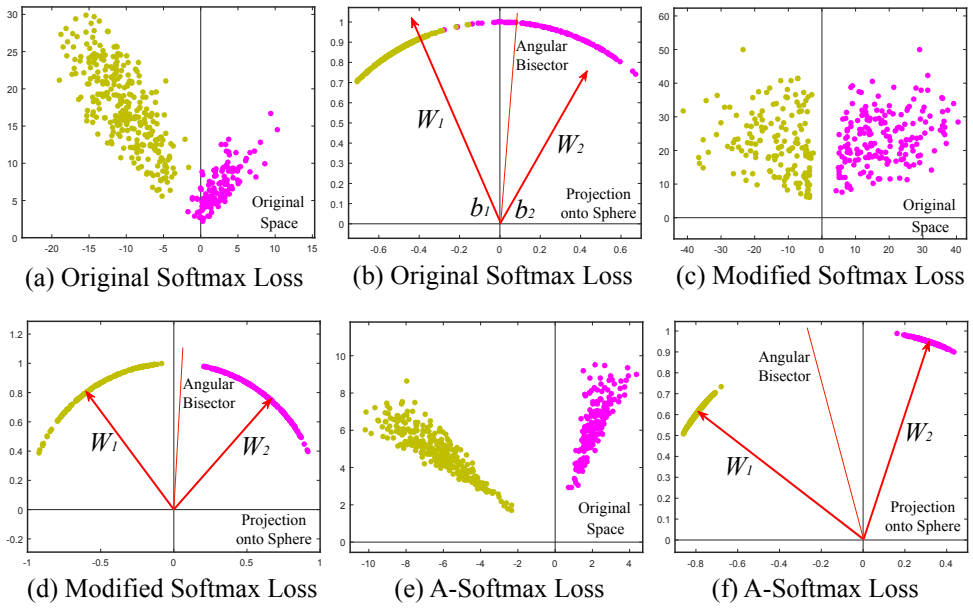


Figure 2.6: Comparison of open-set and closed-set face recognition.

features.

Desired features for open-set FR are expected to satisfy the criterion that the maximal intra-class distance is smaller than the minimal inter-class distance under a certain metric space. This criterion is necessary if we want to achieve perfect accuracy using nearest neighbor. However, learning features with this criterion is generally difficult because of the intrinsically large intra-class variation and high inter-class similarity [66] that faces exhibit.

Few CNN-based approaches are able to effectively formulate the aforementioned criterion in loss functions. Pioneering work [67, 36] learn face features via the softmax loss, but softmax loss only learns separable features that are not discriminative enough. To address



(a) Original Softmax Loss (b) Original Softmax Loss (c) Modified Softmax Loss

(d) Modified Softmax Loss (e) A-Softmax Loss (f) A-Softmax Loss

Figure 2.7: Comparison among softmax loss, modified softmax loss and A-Softmax loss. In this toy experiment, we construct a CNN to learn 2-D features on a subset of the CASIA face dataset. In specific, we set the output dimension of FC1 layer as 2 and visualize the learned features. Yellow dots represent the first class face features, while purple dots represent the second class face features. One can see that features learned by the original softmax loss can not be classified simply via angles, while modified softmax loss can. Our A-Softmax loss can further increase the angular margin of learned features.

this, some methods combine softmax loss with contrastive loss [62, 68] or center loss [63] to enhance the discrimination power of features. [34] adopts triplet loss to supervise the embedding learning, leading to state-of-the-art face recognition results. However, center loss only explicitly encourages intra-class compactness. Both contrastive loss [69] and triplet loss [34] can not constrain on each individual sample, and thus require carefully designed pair/triplet mining procedure, which is both time-consuming and performance-sensitive.

It seems to be a widely recognized choice to impose Euclidean margin to learned features, but a question arises: *Is Euclidean margin always suitable for learning discriminative face features?* To answer this question, we first look into how Euclidean margin based losses are applied to FR. Most recent approaches [62, 68, 63] combine Euclidean margin based losses with softmax loss to construct a joint supervision. However, as can be observed from Figure 2.7, the features learned by softmax loss have intrinsic angular

distribution (also verified by [63]). In some sense, Euclidean margin based losses are incompatible with softmax loss, so it is not well motivated to combine these two type of losses.

We propose to incorporate angular margin instead. We start with a binary-class case to analyze the softmax loss. The decision boundary in softmax loss is $(\boldsymbol{W}_1-\boldsymbol{W}_2)\boldsymbol{x}+b_1-b_2 = 0$, where $\boldsymbol{W}_i$ and b_i are weights and bias[2] in softmax loss, respectively. If we define $\boldsymbol{x}$ as a feature vector and constrain $\|\boldsymbol{W}_1\| = \|\boldsymbol{W}_2\| = 1$ and $b_1 = b_2 = 0$, the decision boundary becomes $\|\boldsymbol{x}\|(\cos(\theta_1) - \cos(\theta_2)) = 0$, where θ_i is the angle between $\boldsymbol{W}_i$ and $\boldsymbol{x}$. The new decision boundary only depends on θ_1 and θ_2. Modified softmax loss is able to directly optimize angles, enabling CNNs to learn angularly distributed features (Figure 2.7).

Compared to original softmax loss, the features learned by modified softmax loss are angularly distributed, but not necessarily more discriminative. To the end, we generalize the modified softmax loss to angular softmax (A-Softmax) loss. Specifically, we introduce an integer m ($m \geq 1$) to quantitatively control the decision boundary. In binary-class case, the decision boundaries for class 1 and class 2 become $\|\boldsymbol{x}\|(\cos(m\theta_1)-\cos(\theta_2))=0$ and $\|\boldsymbol{x}\|(\cos(\theta_1)-\cos(m\theta_2))=0$, respectively. m quantitatively controls the size of angular margin. Furthermore, A-Softmax loss can be easily generalized to multiple classes, similar to softmax loss. By optimizing A-Softmax loss, the decision regions become more separated, simultaneously enlarging the inter-class margin and compressing the intra-class angular distribution.

A-Softmax loss has clear geometric interpretation. Supervised by A-Softmax loss, the learned features construct a discriminative angular distance metric that is equivalent to geodesic distance on a hypersphere manifold. A-Softmax loss can be interpreted as constraining learned features to be discriminative on a hypersphere manifold, which intrinsically matches the prior that face images lie on a manifold [70, 71, 72]. The close connection between A-Softmax loss and hypersphere manifolds makes the learned features more

[2]If not specified, the weights and biases in the following are corresponding to the fully connected layer in the softmax loss.

effective for face recognition. For this reason, we term the learned features as *SphereFace*.

Moreover, A-Softmax loss can quantitatively adjust the angular margin via a parameter m, enabling us to do quantitative analysis. In the light of this, we derive lower bounds for the parameter m to approximate the desired open-set FR criterion that the maximal intra-class distance should be smaller than the minimal inter-class distance.

Our major contributions can be summarized as follows:

(1) We propose A-Softmax loss for CNNs to learn discriminative face features with clear and novel geometric interpretation. The learned features discriminatively span on a hypersphere manifold, which intrinsically matches the prior that faces also lie on a manifold.

(2) We derive lower bounds for m such that A-Softmax loss can approximate the learning task that minimal inter-class distance is larger than maximal intra-class distance.

(3) We are the very first to show the effectiveness of angular margin in FR. Trained on publicly available CASIA dataset [57], *SphereFace* achieves competitive results on several benchmarks, including Labeled Face in the Wild (LFW), Youtube Faces (YTF) and MegaFace Challenge 1.

2.3.2 Revisiting the Softmax Loss

We revisit the softmax loss by looking into the decision criteria of softmax loss. In binary-class case, the posterior probabilities obtained by softmax loss are

$$p_1 = \frac{\exp(\boldsymbol{W}_1^T \boldsymbol{x} + b_1)}{\exp(\boldsymbol{W}_1^T \boldsymbol{x} + b_1) + \exp(\boldsymbol{W}_2^T \boldsymbol{x} + b_2)} \tag{2.14}$$

$$p_2 = \frac{\exp(\boldsymbol{W}_2^T \boldsymbol{x} + b_2)}{\exp(\boldsymbol{W}_1^T \boldsymbol{x} + b_1) + \exp(\boldsymbol{W}_2^T \boldsymbol{x} + b_2)} \tag{2.15}$$

where $\boldsymbol{x}$ is the learned feature vector. $\boldsymbol{W}_i$ and b_i are weights and bias of last fully connected layer corresponding to class i, respectively. The predicted label will be assigned to class 1

if $p_1 > p_2$ and class 2 if $p_1 < p_2$. By comparing p_1 and p_2, it is clear that $\boldsymbol{W}_1^T\boldsymbol{x} + b_1$ and $\boldsymbol{W}_2^T\boldsymbol{x}+b_2$ determine the classification result. The decision boundary is $(\boldsymbol{W}_1-\boldsymbol{W}_2)\boldsymbol{x}+b_1-b_2 = 0$. We then rewrite $\boldsymbol{W}_i^T\boldsymbol{x}+b_i$ as $\|\boldsymbol{W}_i^T\|\|\boldsymbol{x}\|\cos(\theta_i)+b_i$ where θ_i is the angle between $\boldsymbol{W}_i$ and $\boldsymbol{x}$. Notice that if we normalize the weights and zero the biases ($\|\boldsymbol{W}_i\| = 1$, b_i=0), the posterior probabilities become p_1=$\|\boldsymbol{x}\|\cos(\theta_1)$ and p_2=$\|\boldsymbol{x}\|\cos(\theta_2)$. Note that p_1 and p_2 share the same $\boldsymbol{x}$, the final result only depends on the angles θ_1 and θ_2. The decision boundary also becomes $\cos(\theta_1)-\cos(\theta_2)$=0 (i.e. angular bisector of vector $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$). Although the above analysis is built on binary-calss case, it is trivial to generalize the analysis to multi-class case. During training, the modified softmax loss ($\|\boldsymbol{W}_i\|$=1, b_i=0) encourages features from the i-th class to have smaller angle θ_i (larger cosine distance) than others, which makes angles between $\boldsymbol{W}_i$ and features a reliable metric for classification.

To give a formal expression for the modified softmax loss, we first define the input feature $\boldsymbol{x}_i$ and its label y_i. The original softmax loss can be written as

$$L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -\log\Big(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\Big) \tag{2.16}$$

where f_j denotes the j-th element ($j \in [1, K]$, K is the class number) of the class score vector $\boldsymbol{f}$, and N is the number of training samples. In CNNs, $\boldsymbol{f}$ is usually the output of a fully connected layer $\boldsymbol{W}$, so $f_j = \boldsymbol{W}_j^T\boldsymbol{x}_i + b_j$ and $f_{y_i} = \boldsymbol{W}_{y_i}^T\boldsymbol{x}_i + b_{y_i}$ where $\boldsymbol{x}_i$, $\boldsymbol{W}_j, \boldsymbol{W}_{y_i}$ are the i-th training sample, the j-th and y_i-th column of $\boldsymbol{W}$ respectively. We further reformulate L_i in Equation 2.16 as

$$L_i = -\log\Big(\frac{e^{\boldsymbol{W}_{y_i}^T\boldsymbol{x}_i+b_{y_i}}}{\sum_j e^{\boldsymbol{W}_j^T\boldsymbol{x}_i+b_j}}\Big) = -\log\Big(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\cos(\theta_{y_i,i})+b_{y_i}}}{\sum_j e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_{j,i})+b_j}}\Big) \tag{2.17}$$

in which $\theta_{j,i}(0 \le \theta_{j,i} \le \pi)$ is the angle between vector $\boldsymbol{W}_j$ and $\boldsymbol{x}_i$. As analyzed above, we first normalize $\|\boldsymbol{W}_j\| = 1, \forall j$ in each iteration and zero the biases. Then we have the

modified softmax loss:

$$L_{\text{modified}} = \frac{1}{N}\sum_{i} -\log\left(\frac{e^{\|\boldsymbol{x}_i\|\cos(\theta_{y_i,i})}}{\sum_j e^{\|\boldsymbol{x}_i\|\cos(\theta_{j,i})}}\right) \tag{2.18}$$

Although we can learn features with angular boundary with the modified softmax loss, these features are still not necessarily discriminative. Since we use angles as the distance metric, it is natural to incorporate angular margin to learned features in order to enhance the discrimination power. To this end, we propose a novel way to combine angular margin.

2.3.3 Introducing Angular Margin to Softmax Loss

Instead of designing a new type of loss function and constructing a weighted combination with softmax loss (similar to contrastive loss) , we propose a more natural way to learn angular margin. From the previous analysis of softmax loss, we learn that decision boundaries can greatly affect the feature distribution, so our basic idea is to manipulate decision boundaries to produce angular margin. We first give a motivating binary-class example to explain how our idea works.

Assume a learned feature $\boldsymbol{x}$ from class 1 is given and θ_i is the angle between $\boldsymbol{x}$ and $\boldsymbol{W}_i$, it is known that the modified softmax loss requires $\cos(\theta_1) > \cos(\theta_2)$ to correctly classify $\boldsymbol{x}$. But what if we instead require $\cos(m\theta_1) > \cos(\theta_2)$ where $m \geq 2$ is a integer in order to correctly classify $\boldsymbol{x}$? It is essentially making the decision more stringent than previous, because we require a lower bound[3] of $\cos(\theta_1)$ to be larger than $\cos(\theta_2)$. The decision boundary for class 1 is $\cos(m\theta_1) = \cos(\theta_2)$. Similarly, if we require $\cos(m\theta_2) > \cos(\theta_1)$ to correctly classify features from class 2, the decision boundary for class 2 is $\cos(m\theta_2) = \cos(\theta_1)$. Suppose all training samples are correctly classified, such decision boundaries will produce an angular margin of $\frac{m-1}{m+1}\theta_2^1$ where θ_2^1 is the angle between $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. From angular perspective, correctly classifying $\boldsymbol{x}$ from identity 1 requires $\theta_1 < \frac{\theta_2}{m}$, while correctly classifying $\boldsymbol{x}$ from identity 2 requires $\theta_2 < \frac{\theta_1}{m}$. Both are more difficult

[3]The inequality $\cos(\theta_1) > \cos(m\theta_1)$ holds while $\theta_1 \in [0, \frac{\pi}{m}], m \geq 2$.

Table 2.6: Comparison of decision boundaries in binary case. Note that, θ_i is the angle between $\boldsymbol{W}_i$ and $\boldsymbol{x}$.

Loss Function	Decision Boundary
Softmax Loss	$(\boldsymbol{W}_1 - \boldsymbol{W}_2)\boldsymbol{x} + b_1 - b_2 = 0$
Modified Softmax Loss	$\|\boldsymbol{x}\|(\cos\theta_1 - \cos\theta_2) = 0$
A-Softmax Loss	$\|\boldsymbol{x}\|(\cos m\theta_1 - \cos\theta_2) = 0$ for class 1 $\|\boldsymbol{x}\|(\cos\theta_1 - \cos m\theta_2) = 0$ for class 2

than original $\theta_1 < \theta_2$ and $\theta_2 < \theta_1$, respectively. By directly formulating this idea into the modified softmax loss Equation 2.18, we have

$$L_{\text{ang}} = \frac{1}{N}\sum_i -\log\Big(\frac{e^{\|\boldsymbol{x}_i\|\cos(m\theta_{y_i,i})}}{e^{\|\boldsymbol{x}_i\|\cos(m\theta_{y_i,i})} + \sum_{j\neq y_i} e^{\|\boldsymbol{x}_i\|\cos(\theta_{j,i})}}\Big) \tag{2.19}$$

where $\theta_{y_i,i}$ has to be in the range of $[0, \frac{\pi}{m}]$. In order to get rid of this restriction and make it optimizable in CNNs, we expand the definition range of $\cos(\theta_{y_i,i})$ by generalizing it to a monotonically decreasing angle function $\psi(\theta_{y_i,i})$ which should be equal to $\cos(\theta_{y_i,i})$ in $[0, \frac{\pi}{m}]$. Therefore, our proposed A-Softmax loss is formulated as:

$$L_{\text{ang}} = \frac{1}{N}\sum_i -\log\Big(\frac{e^{\|\boldsymbol{x}_i\|\psi(\theta_{y_i,i})}}{e^{\|\boldsymbol{x}_i\|\psi(\theta_{y_i,i})} + \sum_{j\neq y_i} e^{\|\boldsymbol{x}_i\|\cos(\theta_{j,i})}}\Big) \tag{2.20}$$

in which we define $\psi(\theta_{y_i,i}) = (-1)^k\cos(m\theta_{y_i,i}) - 2k$, $\theta_{y_i,i} \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]$ and $k \in [0, m-1]$. $m \geq 1$ is an integer that controls the size of angular margin. When $m = 1$, it becomes the modified softmax loss.

The justification of A-Softmax loss can also be made from decision boundary perspective. A-Softmax loss adopts different decision boundary for different class (each boundary is more stringent than the original), thus producing angular margin. The comparison of decision boundaries is given in Table 2.6. From original softmax loss to modified softmax loss, it is from optimizing inner product to optimizing angles. From modified softmax loss to A-Softmax loss, it makes the decision boundary more stringent and separated. The angular margin increases with larger m and be zero if $m = 1$.

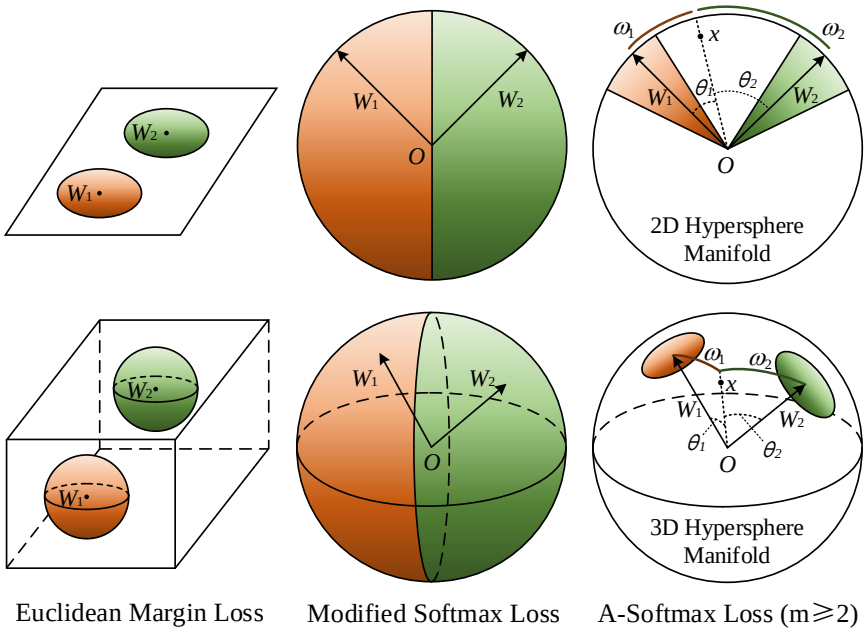


Figure 2.8: Geometry Interpretation of Euclidean margin loss (e.g. contrastive loss, triplet loss, center loss, etc.), modified softmax loss and A-Softmax loss. The first row is 2D feature constraint, and the second row is 3D feature constraint. The orange region indicates the discriminative constraint for class 1, while the green region is for class 2.

Supervised by A-Softmax loss, CNNs learn face features with geometrically interpretable angular margin. Because A-Softmax loss requires $\boldsymbol{W}_i = 1, b_i = 0$, it makes the prediction only depends on angles between the sample $\boldsymbol{x}$ and $\boldsymbol{W}_i$. So $\boldsymbol{x}$ can be classified to the identity with smallest angle. The parameter m is added for the purpose of learning an angular margin between different identities.

To facilitate gradient computation and back propagation, we replace $\cos(\theta_{j,i})$ and $\cos(m\theta_{y_i,i})$ with the expressions only containing $\boldsymbol{W}$ and $\boldsymbol{x}_i$, which is easily done by definition of cosine and multi-angle formula (also the reason why we need m to be an integer). Without θ, we can compute derivative with respect to $\boldsymbol{x}$ and $\boldsymbol{W}$, similar to softmax loss.

2.3.4 Hypersphere Interpretation of A-Softmax Loss

A-Softmax loss has stronger requirements for a correct classification when $m \geq 2$, which generates an angular classification margin between learned features of different classes. A-Softmax loss not only imposes discriminative power to the learned features

via angular margin, but also renders nice and novel hypersphere interpretation. As shown in Figure 2.8, A-Softmax loss is equivalent to learning features that are discriminative on a hypersphere manifold, while Euclidean margin losses learn features in Euclidean space.

To simplify, We take the binary case to analyze the hypersphere interpretation. Considering a sample $\boldsymbol{x}$ from class 1 and two column weights $\boldsymbol{W}_1, \boldsymbol{W}_2$, the classification rule for A-Softmax loss is $\cos(m\theta_1) > \cos(\theta_2)$, equivalently $m\theta_1 < \theta_2$. Notice that θ_1, θ_2 are equal to their corresponding arc length ω_1, ω_2[4] on unit hypersphere $\{v_j, \forall j | \sum_j v_j^2{=}1, v{\geq}0\}$. Because $\|\boldsymbol{W}\|_1 = \|\boldsymbol{W}\|_2 = 1$, the decision replies on the arc length ω_1 and ω_2. The decision boundary is equivalent to $m\omega_1 = \omega_2$, and the constrained region for correctly classifying $\boldsymbol{x}$ to class 1 is $m\omega_1 < \omega_2$. Geometrically speaking, this is a hypercircle-like region lying on a hypersphere manifold. For example, it is a circle-like region on the unit sphere in 3D case, as illustrated in Figure 2.8. Note that larger m leads to smaller hypercircle-like region for each class, which is an explicit discriminative constraint on a manifold. For better understanding, Figure 2.8 provides 2D and 3D visualizations. One can see that A-Softmax loss imposes arc length constraint on a unit circle in 2D case and circle-like region constraint on a unit sphere in 3D case. Our analysis shows that optimizing angles with A-Softmax loss essentially makes the learned features more discriminative on a hypersphere.

2.3.5 Properties of A-Softmax Loss

Property 1. *A-Softmax loss defines a large angular margin learning task with adjustable difficulty. With larger m, the angular margin becomes larger, the constrained region on the manifold becomes smaller, and the corresponding learning task also becomes more difficult.*

We know that the larger m is, the larger angular margin A-Softmax loss constrains. There exists a minimal m that constrains the maximal intra-class angular distance to be

[4] ω_i is the shortest arc length (geodesic distance) between $\boldsymbol{W}_i$ and the projected point of sample $\boldsymbol{x}$ on the unit hypersphere, while the corresponding θ_i is the angle between $\boldsymbol{W}_i$ and $\boldsymbol{x}$.

smaller than the minimal inter-class angular distance, which can also be observed in our experiments.

Definition 1 (minimal m for desired feature distribution)**.** *m_{min} is the minimal value such that while $m > m_{\text{min}}$, A-Softmax loss defines a learning task where the maximal intra-class angular feature distance is constrained to be smaller than the minimal inter-class angular feature distance.*

Property 2 (lower bound of m_{min} in binary-class case)**.** *In binary-class case, we have $m_{\text{min}} \geq 2 + \sqrt{3}$.*

Proof. We consider the space spaned by $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. Because $m \geq 2$, it is easy to obtain the maximal angle that class 1 spans is $\frac{\theta_{12}}{m-1} + \frac{\theta_{12}}{m+1}$ where θ_{12} is the angle between $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. To require the maximal intra-class feature angular distance smaller than the minimal inter-class feature angular distance, we need to constrain

$$\underbrace{\frac{\theta_{12}}{m-1} + \frac{\theta_{12}}{m+1}}_{\text{max intra-class angle}} \leq \underbrace{\frac{(m-1)\theta_{12}}{m+1}}_{\text{min inter-class angle}}, \quad \theta_{12} \leq \frac{m-1}{m}\pi \tag{2.21}$$

$$\underbrace{\frac{2\pi - \theta_{12}}{m+1} + \frac{\theta_{12}}{m+1}}_{\text{max intra-class angle}} \leq \underbrace{\frac{(m-1)\theta_{12}}{m+1}}_{\text{min inter-class angle}}, \quad \theta_{12} > \frac{m-1}{m}\pi \tag{2.22}$$

After solving these two inequalities, we could have $m_{\text{min}} \geq 2 + \sqrt{3}$, which is a lower bound for binary case. □

Property 3 (lower bound of m_{min} in multi-class case)**.** *Under the assumption that $\boldsymbol{W}_i, \forall i$ are uniformly spaced in the Euclidean space, we have $m_{\text{min}} \geq 3$.*

Proof. We consider the 2D k-class ($k \geq 3$) scenario for the lower bound. Because $\boldsymbol{W}_i, \forall i$ are uniformly spaced in the 2D Euclidean space, we have $\theta_i^{i+1} = \frac{2\pi}{k}$ where θ_i^{i+1} is the angle between $\boldsymbol{W}_i$ and $\boldsymbol{W}_{i+1}$. Since $\boldsymbol{W}_i, \forall i$ are symmetric, we only need to analyze one of them.

Table 2.7: Our CNN architectures with different convolutional layers. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers and residual units are shown in double-column brackets. E.g., [3×3, 64]×4 denotes 4 cascaded convolution layers with 64 filters of size 3×3, and S2 denotes stride 2. FC1 is the fully connected layer.

Layer	4-layer CNN	10-layer CNN	20-layer CNN	36-layer CNN	64-layer CNN
Conv1.x	[3×3, 64]×1, S2	[3×3, 64]×1, S2	[3×3, 64]×1, S2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 1$	[3×3, 64]×1, S2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	[3×3, 64]×1, S2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$
Conv2.x	[3×3, 128]×1, S2	[3×3, 128]×1, S2 $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 1$	[3×3, 128]×1, S2 $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	[3×3, 128]×1, S2 $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	[3×3, 128]×1, S2 $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 8$
Conv3.x	[3×3, 256]×1, S2	[3×3, 256]×1, S2 $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	[3×3, 256]×1, S2 $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 4$	[3×3, 256]×1, S2 $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 8$	[3×3, 256]×1, S2 $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 16$
Conv4.x	[3×3, 512]×1, S2	[3×3, 512]×1, S2	[3×3, 512]×1, S2 $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 1$	[3×3, 512]×1, S2 $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	[3×3, 512]×1, S2 $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$
FC1	512	512	512	512	512

For the i-th class ($\boldsymbol{W}_i$), We need to constrain

$$\underbrace{\frac{\theta_i^{i+1}}{m+1} + \frac{\theta_{i-1}^{i}}{m+1}}_{\text{max intra-class angle}} \leq \underbrace{\min\left\{\frac{(m-1)\theta_i^{i+1}}{m+1}, \frac{(m-1)\theta_{i-1}^{i}}{m+1}\right\}}_{\text{min inter-class angle}} \tag{2.23}$$

After solving this inequality, we obtain $m_{\text{min}} \geq 3$, which is a lower bound for multi-class case. □

Based on this, we use $m = 4$ to approximate the desired feature distribution criteria. Since the lower bounds are not necessarily tight, giving a tighter lower bound and a upper bound under certain conditions is also possible, which we leave to the future work. Experiments also show that larger m consistently works better and $m = 4$ will usually suffice.

2.3.6 Experiments

Experimental Settings

Preprocessing. We only use standard preprocessing. The face landmarks in all images are detected by MTCNN [73]. The cropped faces are obtained by similarity transformation. Each pixel ($[0, 255]$) in RGB images is normalized by subtracting 127.5 and then being divided by 128.

CNNs Setup. Caffe [43] is used to implement A-Softmax loss and CNNs. The general framework to train and extract *SphereFace* features is shown in Figure 2.9. We use residual units [61] in our CNN architecture. For fairness, all compared methods use the same CNN architecture (including residual units) as *SphereFace*. CNNs with different depths (4, 10, 20, 36, 64) are used to better evaluate our method. The specific settings for diffffernt CNNs we used are given in Table 2.7. According to the analysis in subsection 2.3.5, we usually set m as 4 in A-Softmax loss unless specified. These models are trained with batch size of 128 on four GPUs. The learning rate begins with 0.1 and is divided by 10 at the 16K, 24K iterations. The training is finished at 28K iterations.

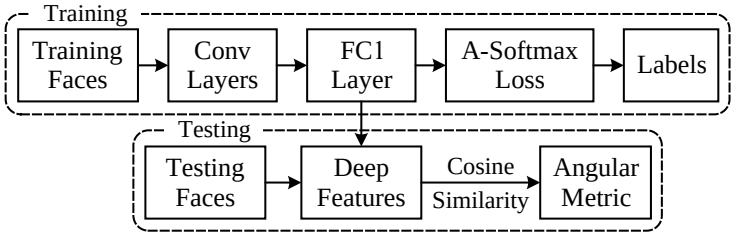


Figure 2.9: Training and Extracting *SphereFace* features.

Training Data. We use publicly available web-collected training dataset CASIA-WebFace [57] (after excluding the images of identities appearing in testing sets) to train our CNN models. CASIA-WebFace has 494,414 face images belonging to 10,575 different individuals. These face images are horizontally flipped for data augmentation. Notice that the scale of our training data (0.49M) is relatively small, especially compared to other private datasets used in DeepFace [67] (4M), VGGFace [55] (2M) and FaceNet [34] (200M).

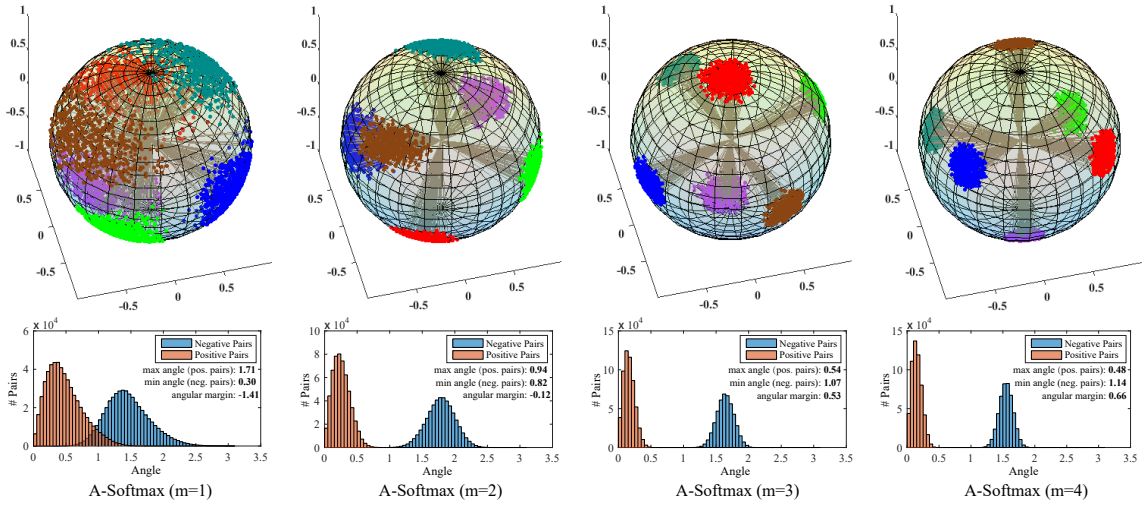


Figure 2.10: Visualization of features learned with different m. The first row shows the 3D features projected on the unit sphere. The projected points are the intersection points of the feature vectors and the unit sphere. The second row shows the angle distribution of both positive pairs and negative pairs (we choose class 1 and class 2 from the subset to construct positive and negative pairs). Orange area indicates positive pairs while blue indicates negative pairs. All angles are represented in radian. Note that, this visualization experiment uses a 6-class subset of the CASIA-WebFace dataset.

Testing. We extract the deep features (*SphereFace*) from the output of the FC1 layer. For all experiments, the final representation of a testing face is obtained by concatenating its original face features and its horizontally flipped features. The score (metric) is computed by the cosine distance of two features. The nearest neighbor classifier and thresholding are used for face identification and verification, respectively.

Exploratory Experiments

Effect of m. To show that larger m leads to larger angular margin (i.e. more discriminative feature distribution on manifold), we perform a toy example with different m. We train A-Softmax loss with 6 individuals that have the most samples in CASIA-WebFace. We set the output feature dimension (FC1) as 3 and visualize the training samples in Figure 2.10. One can observe that larger m leads to more discriminative distribution on the sphere and also larger angular margin, as expected. We also use class 1 (blue) and class 2 (dark green) to construct positive and negative pairs to evaluate the angle distribution

Table 2.8: Accuracy(%) comparison of different m (A-Softmax loss) and original softmax loss on LFW and YTF dataset.

Dataset	Original	m=1	m=2	m=3	m=4
LFW	97.88	97.90	98.40	99.25	**99.42**
YTF	93.1	93.2	93.8	94.4	**95.0**

of features from the same class and different classes. The angle distribution of positive and negative pairs (the second row of Figure 2.10) quantitatively shows the angular margin becomes larger while m increases and every class also becomes more distinct with each other.

Besides visual comparison, we also perform face recognition on LFW and YTF to evaluate the effect of m. For fair comparison, we use 64-layer CNN [10] for all losses. Results are given in Table 2.8. One can observe that while m becomes larger, the accuracy of A-Softmax loss also becomes better, which shows that larger angular margin can bring stronger discrimination power.

Effect of CNN architectures. We train A-Softmax loss ($m = 4$) and original softmax loss with different number of convolution layers. Specific CNN architectures can be found in [10]. From Figure Figure 2.11, one can observe that A-Softmax loss consistently outperforms CNNs with softmax loss (1.54%$\sim$1.91%), indicating that A-Softmax loss is more suitable for open-set FR. Besides, the difficult learning task defined by A-Softmax loss makes full use of the superior learning capability of deeper architectures. A-Softmax loss greatly improve the verification accuracy from 98.20% to 99.42% on LFW, and from 93.4% to 95.0% on YTF. On the contrary, the improvement of deeper standard CNNs is unsatisfactory and also easily get saturated (from 96.60% to 97.75% on LFW, from 91.1% to 93.1% on YTF).

Experiments on LFW and YTF

LFW dataset [41] includes 13,233 face images from 5749 different identities, and YTF dataset [75] includes 3,424 videos from 1,595 different individuals. Both datasets contains

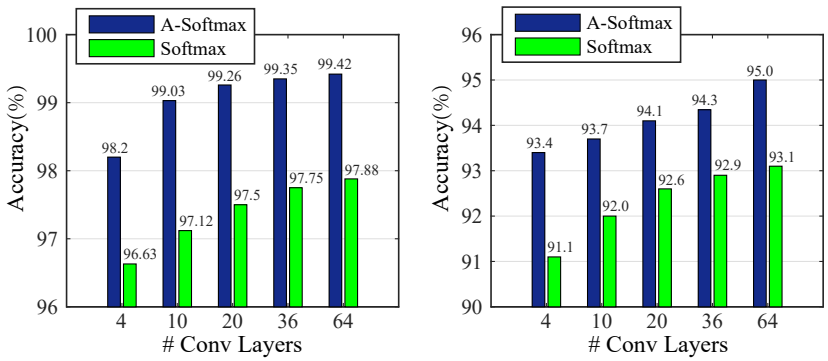


Figure 2.11: Accuracy (%) on LFW and YTF with different number of convolutional layers. Left side is for LFW, while right side is for YTF.

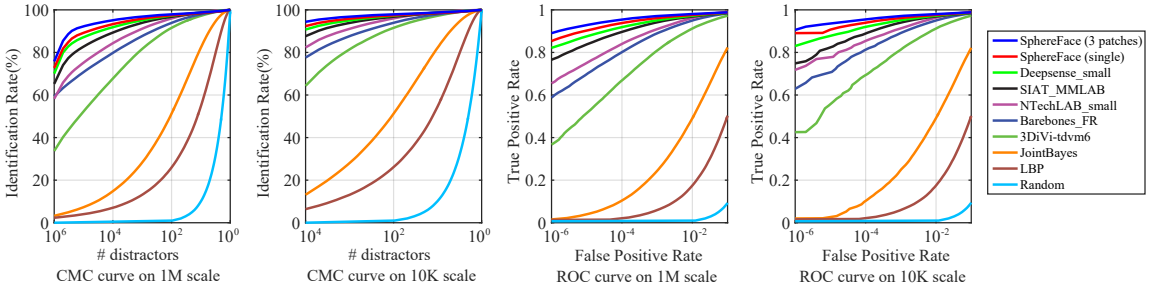


Figure 2.12: CMC and ROC curves of different methods under the small training set protocol.

faces with large variations in pose, expression and illuminations. We follow the unrestricted with labeled outside data protocol [64] on both datasets. The performance of *SphereFace* are evaluated on 6,000 face pairs from LFW and 5,000 video pairs from YTF. The results are given in Table 2.9. For contrastive loss and center loss, we follow the FR convention to form a weighted combination with softmax loss. The weights are selected via cross validation on training set. For L-Softmax [9], we also use $m = 4$. All the compared loss functions share the same 64-layer CNN architecture.

Most of the existing face verification systems achieve high performance with huge training data or model ensemble. While using single model trained on publicly available dataset (CAISA-WebFace, relatively small and having noisy labels), *SphereFace* achieves 99.42% and 95.0% accuracies on LFW and YTF datasets. It is the current best performance trained on WebFace and considerably better than the other models trained on the same dataset.

Table 2.9: Accuracy (%) on LFW and YTF dataset. * denotes the outside data is private (not publicly available). For fair comparison, all loss functions (including ours) we implemented use 64-layer CNN architecture in [10]

Method	Models	Data	LFW	YTF
DeepFace [67]	3	4M*	97.35	91.4
FaceNet [34]	1	200M*	**99.65**	95.1
Deep FR [55]	1	2.6M	98.95	**97.3**
DeepID2+ [56]	1	300K*	98.70	N/A
DeepID2+ [56]	25	300K*	99.47	93.2
Baidu [74]	1	1.3M*	99.13	N/A
Center Face [63]	1	0.7M*	99.28	94.9
Yi et al. [57]	1	WebFace	97.73	92.2
Ding et al. [58]	1	WebFace	98.43	N/A
Liu et al. [9]	1	WebFace	98.71	N/A
Softmax Loss	1	WebFace	97.88	93.1
Softmax+Contrastive [36]	1	WebFace	98.78	93.5
Triplet Loss [34]	1	WebFace	98.70	93.4
L-Softmax Loss [9]	1	WebFace	99.10	94.0
Softmax+Center Loss [63]	1	WebFace	99.05	94.4
SphereFace	1	WebFace	**99.42**	**95.0**

Compared with models trained on high-quality private datasets, *SphereFace* is still very competitive, outperforming most of the existing results in Table 2.9. One should notice that our single model performance is only worse than Google FaceNet which is trained with more than 200 million data.

For fair comparison, we also implement the softmax loss, contrastive loss, center loss, triplet loss, L-Softmax loss [9] and train them with the same 64-layer CNN architecture as A-Softmax loss. As can be observed in Table 2.9, *SphereFace* consistently outperforms the features learned by all these compared losses, showing its superiority in FR tasks.

Experiments on MegaFace Challenge

MegaFace dataset [76] is a recently released testing benchmark with very challenging task to evaluate the performance of face recognition methods at the million scale of distractors. MegaFace dataset contains a gallery set and a probe set. The gallery set contains more than 1 million images from 690K different individuals. The probe set consists of two existing datasets: Facescrub [77] and FGNet. MegaFace has several testing scenarios including

Table 2.10: Performance (%) on MegaFace challenge. "Rank-1 Acc." indicates rank-1 identification accuracy with 1M distractors, and "Ver." indicates verification TAR for 10^{-6} FAR. TAR and FAR denote True Accept Rate and False Accept Rate respectively. For fair comparison, all loss functions (including ours) we implemented use the same deep CNN architecture.

Method	protocol	Rank1 Acc.	Ver.
NTechLAB - facenx large	Large	73.300	85.081
Vocord - DeepVo1	Large	**75.127**	67.318
Deepsense - Large	Large	74.799	**87.764**
Shanghai Tech	Large	74.049	86.369
Google - FaceNet v8	Large	70.496	86.473
Beijing FaceAll_Norm_1600	Large	64.804	67.118
Beijing FaceAll_1600	Large	63.977	63.960
Deepsense - Small	Small	**70.983**	**82.851**
SIAT_MMLAB	Small	65.233	76.720
Barebones FR - cnn	Small	59.363	59.036
NTechLAB - facenx_small	Small	58.218	66.366
3DiVi Company - tdvm6	Small	33.705	36.927
Softmax Loss	Small	54.855	65.925
Softmax+Contrastive Loss [36]	Small	65.219	78.865
Triplet Loss [34]	Small	64.797	78.322
L-Softmax Loss [9]	Small	67.128	80.423
Softmax+Center Loss [63]	Small	65.494	80.146
SphereFace (single model)	Small	**72.729**	**85.561**
SphereFace (3-patch ensemble)	Small	**75.766**	**89.142**

identification, verification and pose invariance under two protocols (large or small training set). The training set is viewed as small if it is less than 0.5M. We evaluate *SphereFace* under the small training set protocol. We adopt two testing protocols: face identification and verification. The results are given in Figure 2.12 and Table 2.10. Note that we use simple 3-patch feature concatenation ensemble as the final performance of *SphereFace*.

Figure 2.12 and Table 2.10 show that *SphereFace* (3 patches ensemble) beats the second best result by a large margins (4.8% for rank-1 identification rate and 6.3% for verification rate) on MegaFace benchmark under the small training dataset protocol. Compared to the models trained on large dataset (500 million for Google and 18 million for NTechLAB), our method still performs better (0.64% for identification rate and 1.4% for verification rate). Moreover, in contrast to their sophisticated network design, we only employ typical CNN architecture supervised by A-Softamx to achieve such excellent performance. For single

model *SphereFace*, the accuracy of face identification and verification are still 72.73% and 85.56% respectively, which already outperforms most state-of-the-art methods. For better evaluation, we also implement the softmax loss, contrastive loss, center loss, triplet loss and L-Softmax loss [9]. Compared to these loss functions trained with the same CNN architecture and dataset, *SphereFace* also shows significant and consistent improvements. These results convincingly demonstrate that the proposed *SphereFace* is well designed for open-set face recognition. One can also see that learning features with large inter-class angular margin can significantly improve the open-set FR performance.

2.4 Concluding Remarks

This chapter has presented two variants (L-Softmax and A-Softmax) of the large-margin softmax loss on hypersphere. These large-margin losses have appealing geometric interpretations and can learn deep features with large angular margin. More importantly, we provide a general framework (Part I:) where one only needs to customize the function $\psi(\theta_{y_i})$ in order to design large-margin losses. To the best of our knowledge, we are the very first to introduce the large angular margin property to the softmax loss and enable deep representation learning on hypersphere. In experiments, we also show that both L-Softmax loss and A-Softmax loss are very effective in visual recognition and face recognition. We believe that they will also be useful in tasks that require large-margin deep features, such as person re-identification and image retrieval.

Part II:

Neural Architectures on Hypersphere

Although the learning objectives discussed in chapter 2 can work well with standard neural architectures, we can better facilitate the deep representation learning on hypersphere by introducing the hyperspherical inductive bias to every layer of the network. To this end, we propose the hyperspherical convolutional networks (SphereNet) [12] that are distinct from conventional inner product based convolutional networks. Specifically, SphereNet is built upon hyperspherical convolutions which give angular representations on hypersphere. Then we generalize SphereNet to a more general decoupled learning framework [20] where the convolution operator is decoupled into a norm-based function and an angle-based function. The resulting neural network is called decoupled network. We have shown that these neural architectures designed for hyperspherical learning yield better generalization, stronger adversarial robustness and faster convergence, compared to their standard convolutional counterparts.

Chapter 3 is based on the following publication:

- *W. Liu*, Y.-M. Zhang, X. Li, Z. Yu, B. Dai, T. Zhao, L. Song. Deep Hyperspherical Learning. *NeurIPS 2017*

Chapter 4 is based on the following publication:

- *W. Liu*, Z. Liu, Z. Yu, B. Dai, R. Lin, Y. Wang, J. Rehg, L. Song. Decoupled Networks. *CVPR 2018*

CHAPTER 3

SPHERENET: HYPERSPHERICAL NEURAL NETWORKS

3.1 Introduction

Recently, deep convolutional neural networks have led to significant breakthroughs on many vision problems such as image classification [1, 42, 60, 2], segmentation [78, 3, 79], object detection [78, 4], etc. While showing stronger representation power over many conventional hand-crafted features, CNNs often require a large amount of training data and face certain training difficulties such as overfitting, vanishing/exploding gradient, covariate shift, etc. The increasing depth of recently proposed CNN architectures have further aggravated the problems.

To address the challenges, regularization techniques such as dropout [1] and orthogonality parameter constraints [28] have been proposed. Batch normalization [44] can also be viewed as an implicit regularization to the network, by normalizing each layer's output distribution. Recently, deep residual learning [2] emerged as a promising way to overcome vanishing gradients in deep networks. However, [80] pointed out that residual networks (ResNets) are essentially an exponential ensembles of shallow networks where they avoid the vanishing/exploding gradient problem but do not provide direct solutions. As a result, training an ultra-deep network still remains an open problem. Besides vanishing/exploding gradient, network optimization is also very sensitive to initialization. Finding better initializations is thus widely studied [38, 81, 82]. In general, having a large parameter space is double-edged considering the benefit of representation power and the associated training difficulties. Therefore, proposing better learning frameworks to overcome such challenges remains important.

In this chapter, we introduce a novel convolutional learning framework that can effec-

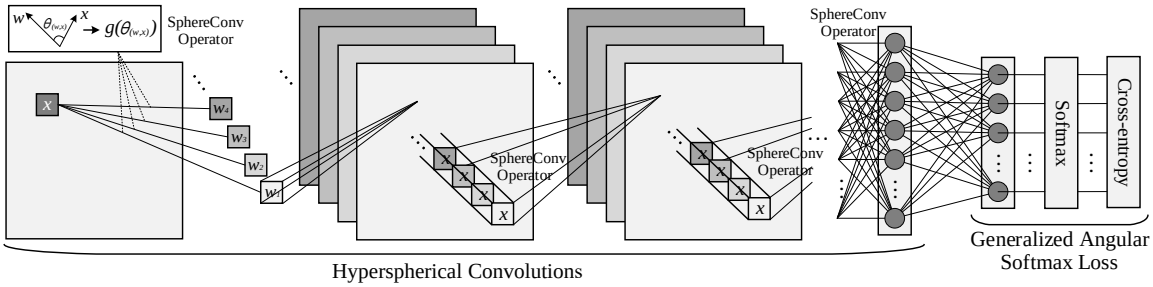


Figure 3.1: Deep hyperspherical convolutional network architecture.

tively alleviate training difficulties, while giving better performance over dot product based convolution. Our idea is to project parameter learning onto unit hyperspheres, where layer activations only depend on the geodesic distance between kernels and input signals[1] instead of their inner products. To this end, we propose the SphereConv operator as the basic module for our network layers. We also propose softmax losses accordingly under such representation framework. Specifically, the proposed softmax losses supervise network learning by also taking the SphereConv activations from the last layer instead of inner products. Note that the geodesic distances on a unit hypersphere is the angles between inputs and kernels. Therefore, the learning objective is essentially a function of the input angles and we call it generalized angular softmax loss in this chapter. The resulting architecture is the hyperspherical convolutional network (SphereNet), which is shown in Figure 3.1.

Our key motivation to propose SphereNet is that angular information matters in convolutional representation learning. We argue this motivation from several aspects: training stability, training efficiency, and generalization power. SphereNet can also be viewed as an implicit regularization to the network by normalizing the activation distributions. The weight norm is no longer important since the entire network operates only on angles. And as a result, the ℓ_2 weight decay is also no longer needed in SphereNet. SphereConv to some extent also alleviates the covariate shift problem [44]. The output of SphereConv operators are bounded from -1 to 1 (0 to 1 if considering ReLU), which makes the variance of each output also bounded.

[1] Without loss of generality, we study CNNs here, but our method is generalizable to any other neural nets.

Our second intuition is that angles preserve the most abundant discriminative information in convolutional learning. We gain such intuition from 2D Fourier transform, where an image is decomposed by the combination of a set of templates with magnitude and phase information in 2D frequency domain. If one reconstructs an image with original magnitudes and random phases, the resulting images are generally not recognizable. However, if one reconstructs the image with random magnitudes and original phases. The resulting images are still recognizable. It shows that the most important structural information in an image for visual recognition is encoded by phases. This fact inspires us to project the network learning into angular space. In terms of low-level information, SphereConv is able to preserve the shape, edge, texture and relative color. SphereConv can learn to selectively drop the color depth but preserve the RGB ratio. Thus the semantic information of an image is preserved.

SphereNet can also be viewed as a non-trivial generalization of [9, 10]. By proposing a loss that discriminatively supervises the network on a hypersphere, [10] achieves state-of-the-art performance on face recognition. However, the rest of the network remains a conventional convolution network. In contrast, SphereNet not only generalizes the hyperspherical constraint to every layer, but also to different nonlinearity functions of input angles. Specifically, we propose three instances of SphereConv operators: linear, cosine and sigmoid. The sigmoid SphereConv is the most flexible one with a parameter controlling the shape of the angular function. As a simple extension to the sigmoid SphereConv, we also present a learnable SphereConv operator. Moreover, the proposed generalized angular softmax (GA-Softmax) loss naturally generalizes the angular supervision in [10] using the SphereConv operators. Additionally, the SphereConv can serve as a normalization method that is comparable to batch normalization, leading to an extension to spherical normalization (SphereNorm).

SphereNet can be easily applied to other network architectures such as GoogLeNet [60], VGG [42] and ResNet [2]. One simply needs to replace the convolutional operators and the

loss functions with the proposed SphereConv operators and hyperspherical loss functions. In summary, SphereConv can be viewed as an alternative to the original convolution operators, and serves as a new measure of correlation. SphereNet may open up an interesting direction to explore the neural networks. We ask the question *whether inner product based convolution operator is an optimal correlation measure for all tasks?* Our answer to this question is likely to be "no".

3.2 Hyperspherical Convolution Operator

3.2.1 Definition

The convolutional operator in CNNs is simply a linear matrix multiplication, written as $\mathcal{F}(\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b_\mathcal{F}$ where $\boldsymbol{w}$ is a convolutional filter, $\boldsymbol{x}$ denotes a local patch from the bottom feature map and $b_\mathcal{F}$ is the bias. The matrix multiplication here essentially computes the similarity between the local patch and the filter. Thus the standard convolution layer can be viewed as patch-wise matrix multiplication. Different from the standard convolutional operator, the hyperspherical convolution (SphereConv) operator computes the similarity on a hypersphere and is defined as:

$$\mathcal{F}_s(\boldsymbol{w}, \boldsymbol{x}) = g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) + b_{\mathcal{F}_s}, \tag{3.1}$$

where $\theta_{(\boldsymbol{w},\boldsymbol{x})}$ is the angle between the kernel parameter $\boldsymbol{w}$ and the local patch $\boldsymbol{x}$. $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ indicates a function of $\theta_{(\boldsymbol{w},\boldsymbol{x})}$ (usually a monotonically decreasing function), and $b_{\mathcal{F}_s}$ is the bias. To simplify analysis and discussion, the bias terms are usually left out. The angle $\theta_{(\boldsymbol{w},\boldsymbol{x})}$ can be interpreted as the geodesic distance (arc length) between $\boldsymbol{w}$ and $\boldsymbol{x}$ on a unit hypersphere. In contrast to the convolutional operator that works in the entire space, SphereConv only focuses on the angles between local patches and the filters, and therefore operates on the hypersphere space. In this chapter, we present three specific instances of the SphereConv Operator. To facilitate the computation, we constrain the output of

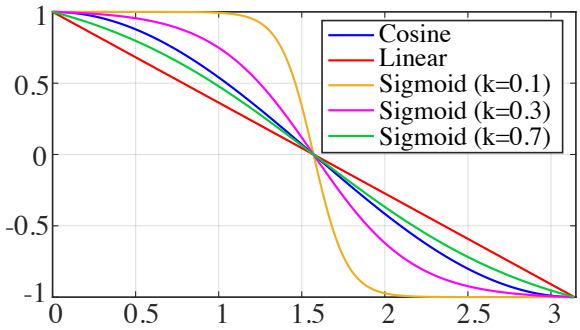


Figure 3.2: SphereConv operators.

SphereConv operators to $[-1, 1]$ (although it is not a necessary requirement).

Linear SphereConv. In linear SphereConv operator, g is a linear function of $\theta_{(\boldsymbol{w},\boldsymbol{x})}$, with the form:

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = a\theta_{(\boldsymbol{w},\boldsymbol{x})} + b, \tag{3.2}$$

where a and b are parameters for the linear SphereConv operator. In order to constrain the output range to $[0, 1]$ while $\theta_{(\boldsymbol{w},\boldsymbol{x})} \in [0, \pi]$, we use $a = -\frac{2}{\pi}$ and $b = 1$ (not necessarily optimal design).

Cosine SphereConv. The cosine SphereConv operator is a nonlinear function of $\theta_{(\boldsymbol{w},\boldsymbol{x})}$, with its g being the form of

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = \cos(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{3.3}$$

which can be reformulated as $\frac{\boldsymbol{w}^T\boldsymbol{x}}{\|\boldsymbol{w}\|_2\|\boldsymbol{x}\|_2}$. Therefore, it can be viewed as a doubly normalized convolutional operator, which bridges the SphereConv operator and convolutional operator.

Sigmoid SphereConv. The Sigmoid SphereConv operator is derived from the Sigmoid function and its g can be written as

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = \frac{1 + \exp(-\frac{\pi}{2k})}{1 - \exp(-\frac{\pi}{2k})} \cdot \frac{1 - \exp\left(\frac{\theta_{(\boldsymbol{w},\boldsymbol{x})}}{k} - \frac{\pi}{2k}\right)}{1 + \exp\left(\frac{\theta_{(\boldsymbol{w},\boldsymbol{x})}}{k} - \frac{\pi}{2k}\right)}, \tag{3.4}$$

where $k > 0$ is the parameter that controls the curvature of the function. While k is close to 0, $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ will approximate the step function. While k becomes larger, $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ is more like a linear function, i.e., the linear SphereConv operator. Sigmoid SphereConv is one

instance of the parametric SphereConv family. With more parameters being introduced, the parametric SphereConv can have richer representation power. To increase the flexibility of the parametric SphereConv, we will discuss the case where these parameters can be jointly learned via back-prop later in the chapter.

3.2.2 Optimization

The optimization of the SphereConv operators is nearly the same as the convolutional operator and also follows the standard back-propagation. Using the chain rule, we have the gradient of the SphereConv with respect to the weights and the feature input:

$$\frac{\partial g(\theta_{(\boldsymbol{w},\boldsymbol{x})})}{\partial \boldsymbol{w}} = \frac{\partial g(\theta_{(\boldsymbol{w},\boldsymbol{x})})}{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}} \cdot \frac{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}}{\partial \boldsymbol{w}}, \quad \frac{\partial g(\theta_{(\boldsymbol{w},\boldsymbol{x})})}{\partial \boldsymbol{x}} = \frac{\partial g(\theta_{(\boldsymbol{w},\boldsymbol{x})})}{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}} \cdot \frac{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}}{\partial \boldsymbol{x}}. \tag{3.5}$$

For different SphereConv operators, both $\frac{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}}{\partial \boldsymbol{w}}$ and $\frac{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}}{\partial \boldsymbol{x}}$ are the same, so the only difference lies in the $\frac{\partial g(\theta_{(\boldsymbol{w},\boldsymbol{x})})}{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}}$ part. For $\frac{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}}{\partial \boldsymbol{w}}$, we have

$$\frac{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}}{\partial \boldsymbol{w}} = \frac{\partial \arccos\left(\frac{\boldsymbol{w}^T\boldsymbol{x}}{\|\boldsymbol{w}\|_2\|\boldsymbol{x}\|_2}\right)}{\partial \boldsymbol{w}}, \quad \frac{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}}{\partial \boldsymbol{x}} = \frac{\partial \arccos\left(\frac{\boldsymbol{w}^T\boldsymbol{x}}{\|\boldsymbol{w}\|_2\|\boldsymbol{x}\|_2}\right)}{\partial \boldsymbol{x}}, \tag{3.6}$$

which are straightforward to compute and therefore neglected here. Because $\frac{\partial g(\theta_{(\boldsymbol{w},\boldsymbol{x})})}{\partial \theta_{(\boldsymbol{w},\boldsymbol{x})}}$ for the linear SphereConv, the cosine SphereConv and the Sigmoid SphereConv are a, $-\sin(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ and $\frac{-2\exp(\theta_{(\boldsymbol{w},\boldsymbol{x})}/k-\pi/2k)}{k(1+\exp(\theta_{(\boldsymbol{w},\boldsymbol{x})}/k-\pi/2k))^2}$ respectively, all these partial gradients can be easily computed.

3.2.3 Theoretical Insights

We provide a fundamental analysis for the cosine SphereConv operator in the case of linear neural network to justify that the SphereConv operator can improve the conditioning of the problem. In specific, we consider one layer of linear neural network, where the observation is $\boldsymbol{F} = \boldsymbol{U}^*\boldsymbol{V}^{*\top}$ (ignore the bias), $\boldsymbol{U}^* \in \mathbb{R}^{n\times k}$ is the weight, and $\boldsymbol{V}^* \in \mathbb{R}^{m\times k}$ is

the input that embeds weights from previous layers. Without loss of generality, we assume the columns satisfying $\|\boldsymbol{U}_{i,:}\|_2 = \|\boldsymbol{V}_{j,:}\|_2 = 1$ for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$, and consider

$$\min_{\boldsymbol{U}\in\mathbb{R}^{n\times k}, \boldsymbol{V}\in\mathbb{R}^{m\times k}} \mathcal{G}(\boldsymbol{U}, \boldsymbol{V}) = \tfrac{1}{2}\|\boldsymbol{F} - \boldsymbol{U}\boldsymbol{V}^\top\|_{\mathrm{F}}^2. \tag{3.7}$$

This is closely related with the matrix factorization and Equation 3.7 can be also viewed as the expected version for the matrix sensing problem [83]. The following lemma demonstrates a critical scaling issue of Equation 3.7 for $\boldsymbol{U}$ and $\boldsymbol{V}$ that significantly deteriorate the conditioning without changing the objective of Equation 3.7.

Lemma 1. *Consider a pair of global optimal points $\boldsymbol{U}, \boldsymbol{V}$ satisfying $\boldsymbol{F} = \boldsymbol{U}\boldsymbol{V}^\top$ and $Tr(\boldsymbol{V}^\top\boldsymbol{V} \otimes \boldsymbol{I}_n) \leq Tr(\boldsymbol{U}^\top\boldsymbol{U} \otimes \boldsymbol{I}_m)$. For any real $c > 1$, let $\tilde{\boldsymbol{U}} = c\boldsymbol{U}$ and $\tilde{\boldsymbol{V}} = \boldsymbol{V}/c$, then we have $\kappa(\nabla^2\mathcal{G}(\tilde{\boldsymbol{U}}, \tilde{\boldsymbol{V}})) = \Omega(c^2\kappa(\nabla^2\mathcal{G}(\boldsymbol{U}, \boldsymbol{V})))$, where $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$ is the restricted condition number with $\lambda_{\max}$ being the largest eigenvalue and $\lambda_{\min}$ being the smallest nonzero eigenvalue.*

Lemma 1 implies that the conditioning of the problem Equation 3.7 at a unbalanced global optimum scaled by a constant c is $\Omega(c^2)$ times larger than the conditioning of the problem at a balanced global optimum. Note that $\lambda_{\min} = 0$ may happen, thus we consider the restricted condition here. Similar results hold beyond global optima. This is an undesired geometric structure, which further leads to slow and unstable optimization procedures, e.g., using stochastic gradient descent (SGD). This motivates us to consider the SphereConv operator discussed above, which is equivalent to projecting data onto the hypersphere and leads to a better conditioned problem.

Next, we consider our proposed cosine SphereConv operator for one-layer of the linear neural network. Based on our previous discussion on SphereConv, we consider an equivalent problem:

$$\min_{\boldsymbol{U}\in\mathbb{R}^{n\times k}, \boldsymbol{V}\in\mathbb{R}^{m\times k}} \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) = \tfrac{1}{2}\|\boldsymbol{F} - \boldsymbol{D}_{\boldsymbol{U}}\boldsymbol{U}\boldsymbol{V}^\top\boldsymbol{D}_{\boldsymbol{V}}\|_{\mathrm{F}}^2, \tag{3.8}$$

where $\boldsymbol{D}_{\boldsymbol{U}} = \mathrm{diag}\big(\frac{1}{\|\boldsymbol{U}_{1,:}\|_2}, \ldots, \frac{1}{\|\boldsymbol{U}_{n,:}\|_2}\big) \in \mathbb{R}^{n\times n}$ and $\boldsymbol{D}_{\boldsymbol{V}} = \mathrm{diag}\big(\frac{1}{\|\boldsymbol{V}_{1,:}\|_2}, \ldots, \frac{1}{\|\boldsymbol{V}_{m,:}\|_2}\big) \in$

$\mathbb{R}^{m\times m}$ are diagonal matrices. We provide an analogous result to Lemma 1 for Equation 3.8.

Lemma 2. *For any real $c > 1$, let $\tilde{\boldsymbol{U}} = c\boldsymbol{U}$ and $\tilde{\boldsymbol{V}} = \boldsymbol{V}/c$, then we have $\lambda_i(\nabla^2\mathcal{G}_S(\tilde{\boldsymbol{U}}, \tilde{\boldsymbol{V}})) = \lambda_i(\nabla^2\mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}))$ for all $i \in [(n+m)k] = \{1, 2, \ldots, (n+m)k\}$ and $\kappa(\nabla^2\mathcal{G}(\tilde{\boldsymbol{U}}, \tilde{\boldsymbol{V}})) = \kappa(\nabla^2\mathcal{G}(\boldsymbol{U}, \boldsymbol{V}))$, where κ is defined as in Lemma 1.*

We have from Lemma 2 that the issue of increasing condition caused by the scaling is eliminated by the SphereConv operator in the entire parameter space. This enhances the geometric structure over Equation 3.7, which further results in improved convergence of optimization procedures. If we extend the result from one layer to multiple layers, the scaling issue propagates. Roughly speaking, when we train N layers, in the worst case, the conditioning of the problem can be c^N times worse with a scaling factor $c > 1$. The analysis is similar to the one layer case, but the computation of the Hessian matrix and associated eigenvalues are much more complicated. Though our analysis is elementary, we provide an important insight and a straightforward illustration of the advantage for using the SphereConv operator. The extension to more general cases, e..g, using nonlinear activation function (e.g., ReLU), requires much more sophisticated analysis to bound the eigenvalues of Hessian for objectives, which is deferred to future investigation.

3.2.4 Discussion

Comparison to convolutional operators. Convolutional operators compute the inner product between the kernels and the local patches, while the SphereConv operators compute a function of the angle between the kernels and local patches. If we normalize the convolutional operator in terms of both $\boldsymbol{w}$ and $\boldsymbol{x}$, then the normalized convolutional operator is equivalent to the cosine SphereConv operator. Essentially, they use different metric spaces. Interestingly, SphereConv operators can also be interpreted as a function of the Geodesic distance on a unit hypersphere.

Extension to fully connected layers. Because the fully connected layers can be viewed as a special convolution layer with the kernel size equal to the input feature map, the

SphereConv operators could be easily generalized to the fully connected layers. It also indicates that SphereConv operators could be used not only to deep CNNs, but also to linear models like logistic regression, SVM, etc.

Network Regularization. Because the norm of weights is no longer crucial, we stop using the ℓ_2 weight decay to regularize the network. SphereNets are learned on hyperspheres, so we regularize the network based on angles instead of norms. To avoid redundant kernels, we want the kernels uniformly spaced around the hypersphere, but it is difficult to formulate such constraints. As a tradeoff, we encourage the orthogonality. Given a set of kernels $\boldsymbol{W}$ where the i-th column $\boldsymbol{W}_i$ is the weights of the i-th kernel, the network will also minimize $\|\boldsymbol{W}^\top\boldsymbol{W}-\boldsymbol{I}\|_F^2$ where $\boldsymbol{I}$ is an identity matrix.

Determining the optimal SphereConv. In practice, we could treat different types of SphereConv as a hyperparameter and use the cross validation to determine which SphereConv is the most suitable one. For sigmoid SphereConv, we could also use the cross validation to determine its hyperparameter k. In general, we need to specify a SphereConv operator before using it, but prefixing a SphereConv may not be an optimal choice (even using cross validation). What if we treat the hyperparameter k in sigmoid SphereConv as a learnable parameter and use the back-prop to learn it? Following this idea, we further extend sigmoid SphereConv to a learnable SphereConv in the next subsection.

SphereConv as normalization. Because SphereConv could partially address the covariate shift, it could also serve as a normalization method similar to batch normalization. Differently, SphereConv normalizes the network in terms of feature map and kernel weights, while batch normalization is for the mini-batches. Thus they do not contradict with each other and can be used simultaneously.

3.2.5 Extension: Learnable SphereConv and SphereNorm

Learnable SphereConv. It is a natural idea to replace the current prefixed SphereConv with a learnable one. There will be plenty of parametrization choices for the SphereConv

to be learnable, and we present a very simple learnable SphereConv operator based on the sigmoid SphereConv. Because the sigmoid SphereConv has a hyperparameter k, we could treat it as a learnable parameter that can be updated by back-prop. In back-prop, k is updated using $k^{t+1} = k^t + \eta \frac{\partial L}{\partial k}$ where t denotes the current iteration index and $\frac{\partial L}{\partial k}$ can be easily computed by the chain rule. Usually, we also require k to be positive. The learning of k is in fact similar to the parameter learning in PReLU [38].

SphereNorm: hyperspherical learning as a normalization method. Similar to batch normalization (BatchNorm), we note that the hyperspherical learning can also be viewed as a way of normalization, because SphereConv constrain the output value in $[-1, 1]$ ($[0, 1]$ after ReLU). Different from BatchNorm, SphereNorm normalizes the network based on spatial information and the weights, so it has nothing to do with the mini-batch statistic. Because SphereNorm normalizes both the input and weights, it could avoid covariate shift due to large weights and large inputs while BatchNorm could only prevent covariate shift caused by the inputs. In such sense, it will work better than BatchNorm when the batch size is small. Besides, SphereConv is more flexible in terms of design choices (e.g. linear, cosine, and sigmoid) and each may lead to different advantages.

Similar to BatchNorm, we could use a rescaling strategy for the SphereNorm. Specifically, we rescale the output of SphereConv via $\beta \mathcal{F}_s(\boldsymbol{w}, \boldsymbol{x}) + \gamma$ where β and γ are learned by back-prop (similar to BatchNorm, the rescaling parameters can be either learned or prefixed). In fact, SphereNorm does not contradict with the BatchNorm at all and can be used simultaneously with BatchNorm. Interestingly, we find using both is empirically better than using either one alone.

3.3 Learning Objective on Hyperspheres

For learning on hyperspheres, we can either use the conventional loss function such as softmax loss, or use some loss functions that are tailored for the SphereConv operators. We present some possible choices for these tailored loss functions.

Weight-normalized Softmax Loss. The input feature and its label are denoted as $\boldsymbol{x}_i$ and y_i, respectively. The original softmax loss can be written as $L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$ where N is the number of training samples and f_j is the score of the j-th class ($j \in [1, K]$, K is the number of classes). The class score vector $\boldsymbol{f}$ is usually the output of a fully connected layer $\boldsymbol{W}$, so we have $f_j = \boldsymbol{W}_j^\top \boldsymbol{x}_i + b_j$ and $f_{y_i} = \boldsymbol{W}_{y_i}^\top \boldsymbol{x}_i + b_{y_i}$ in which $\boldsymbol{x}_i$, $\boldsymbol{W}_j$, and $\boldsymbol{W}_{y_i}$ are the i-th training sample, the j-th and y_i-th column of $\boldsymbol{W}$ respectively. We can rewrite L_i as

$$L_i = -\log\left(\frac{e^{\boldsymbol{W}_{y_i}^\top \boldsymbol{x}_i + b_{y_i}}}{\sum_j e^{\boldsymbol{W}_j^\top \boldsymbol{x}_i + b_j}}\right) = -\log\left(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\cos(\theta_{y_i,i}) + b_{y_i}}}{\sum_j e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_{j,i}) + b_j}}\right), \tag{3.9}$$

where $\theta_{j,i} (0 \leq \theta_{j,i} \leq \pi)$ is the angle between vector $\boldsymbol{W}_j$ and $\boldsymbol{x}_i$. The decision boundary of the original softmax loss is determined by the vector $\boldsymbol{f}$. Specifically in the binary-class case, the decision boundary of the softmax loss is $\boldsymbol{W}_1^\top \boldsymbol{x} + b_1 = \boldsymbol{W}_2^\top \boldsymbol{x} + b_2$. Considering the intuition of the SphereConv operators, we want to make the decision boundary only depend on the angles. To this end, we normalize the weights ($\|\boldsymbol{W}_j\| = 1$) and zero out the biases ($b_j = 0$), following the intuition in [10] (sometimes we could keep the biases while data is imbalanced). The decision boundary becomes $\|\boldsymbol{x}\|\cos(\theta_1) = \|\boldsymbol{x}\|\cos(\theta_2)$. Similar to SphereConv, we could generalize the decision boundary to $\|\boldsymbol{x}\| g(\theta_1) = \|\boldsymbol{x}\| g(\theta_2)$, so the weight-normalized softmax (W-Softmax) loss can be written as

$$L_i = -\log\left(\frac{e^{\|\boldsymbol{x}_i\| g(\theta_{y_i,i})}}{\sum_j e^{\|\boldsymbol{x}_i\| g(\theta_{j,i})}}\right), \tag{3.10}$$

where $g(\cdot)$ can take the form of linear SphereConv, cosine SphereConv, or sigmoid SphereConv. Thus we also term these three difference weight-normalized loss functions as linear W-Softmax loss, cosine W-Softmax loss, and sigmoid W-Softmax loss, respectively.

Generalized Angular Softmax Loss. Inspired by [10], we use a multiplicative parameter m to impose margins on hyperspheres. We propose a generalized angular softmax (GA-Softmax) loss which extends the W-Softmax loss to a loss function that favors large

angular margin feature distribution. In general, the GA-Softmax loss is formulated as

$$L_i = -\log\left(\frac{e^{\|\boldsymbol{x}_i\| g(m\theta_{y_i,i})}}{e^{\|\boldsymbol{x}_i\| g(m\theta_{y_i,i})} + \sum_{j\neq y_i} e^{\|\boldsymbol{x}_i\| g(\theta_{j,i})}}\right), \tag{3.11}$$

where $g(\cdot)$ could also have the linear, cosine and sigmoid form, similar to the W-Softmax loss. We can see A-Softmax loss [10] is exactly the cosine GA-Softmax loss and W-Softmax loss is the special case ($m = 1$) of GA-Sofmtax loss. Note that we usually require $\theta_{j,i} \in [0, \frac{\pi}{m}]$, because $\cos(\theta_{j,i})$ is only monotonically decreasing in $[0, \pi]$. To address this, [9, 10] construct a monotonically decreasing function recursively using the $[0, \frac{\pi}{m}]$ part of $\cos(m\theta_{j,i})$. Although it indeed partially addressed the issue, it may introduce a number of saddle points (w.r.t. $\boldsymbol{W}$) in the loss surfaces. Originally, $\frac{\partial g}{\partial \theta}$ will be close to 0 only when θ is close to 0 and π. However, in L-Softmax [9] or A-Softmax (cosine GA-Softmax), it is not the case. $\frac{\partial g}{\partial \theta}$ will be 0 when $\theta = \frac{k\theta}{m}, k = 0, \cdots, m$. It will possibly cause instability in training. The sigmoid GA-Softmax loss also has similar issues. However, if we use the linear GA-Softmax loss, this problem will be automatically solved and the training will possibly become more stable in practice. There will also be a lot of choices of $g(\cdot)$ to design a specific GA-Sofmtax loss, and each one has different optimization dynamics. The optimal one may depend on the task itself (e.g. cosine GA-Softmax has been shown effective in deep face recognition [10]).

Discussion of Sphere-normalized Softmax Loss. We have also considered the sphere-normalized softmax loss (S-Softmax), which simultaneously normalizes the weights ($\boldsymbol{W}_j$) and the feature $\boldsymbol{x}$. It seems to be a more natural choice than W-Softmax for the proposed SphereConv and makes the entire framework more unified. In fact, we have tried this and the empirical results are not that good, because the optimization seems to become very difficult. If we use the S-Softmax loss to train a network from scratch, we can not get reasonable results without using extra tricks, which is the reason we do not use it in this chapter. For completeness, we give some discussions here. Normally, it is very difficult

to make the S-Softmax loss value to be small enough, because we normalize the features to unit hypersphere. To make this loss work, we need to either normalize the feature to a value much larger than 1 (hypersphere with large radius) and then tune the learning rate or first train the network with the softmax loss from scratch and then use the S-Softmax loss for finetuning.

3.4 Experiments and Results

3.4.1 Experimental Settings

We will first perform comprehensive ablation study and exploratory experiments for the proposed SphereNets, and then evaluate the SphereNets on image classification. For the image classification task, we perform experiments on CIFAR-10 (only with random left-right flipping), CIFAR-10+ (with full data augmentation), CIFAR-100 and large-scale Imagenet 2012 datasets [84].

General Settings. For CIFAR-10, CIFAR-10+ and CIFAR-100, we follow the same settings from [85, 9]. For Imagenet 2012 dataset, we mostly follow the settings in [1]. We attach more details in section B.2. For fairness, batch normalization and ReLU are used in all methods if not specified. All the comparisons are made to be fair. Compared CNNs have the same architecture with SphereNets.

Training. section B.1 gives the network details. For CIFAR-10 and CIFAR-100, we use the ADAM, starting with the learning rate 0.001. The batch size is 128 if not specified. The learning rate is divided by 10 at 34K, 54K iterations and the training stops at 64K. For both A-Softmax and GA-Softmax loss, we use $m=4$. For Imagenet-2012, we use the SGD with momentum 0.9. The learning rate starts with 0.1, and is divided by 10 at 200K and 375K iterations. The training stops at 550K iteration.

3.4.2 Ablation Study and Exploratory Experiments

We perform comprehensive Ablation and exploratory study on the SphereNet and evaluate every component individually in order to analyze its advantages. We use the 9-layer CNN as default (if not specified) and perform the image classification on CIFAR-10 without any data augmentation.

Table 3.1: Classification accuracy (%) with different loss functions.

SphereConv Operator / Loss	Original Softmax	Sigmoid (0.1) W-Softmax	Sigmoid (0.3) W-Softmax	Sigmoid (0.7) W-Softmax	Linear W-Softmax	Cosine W-Softmax	A-Softmax (m=4)	GA-Softmax (m=4)
Sigmoid (0.1)	90.97	90.91	90.89	90.88	91.07	91.13	91.87	91.99
Sigmoid (0.3)	91.08	**91.44**	91.37	91.21	**91.34**	**91.28**	92.13	**92.38**
Sigmoid (0.7)	91.05	91.16	**91.47**	91.07	90.99	91.18	**92.22**	92.36
Linear	**91.10**	90.93	91.42	90.96	90.95	91.24	92.21	92.32
Cosine	90.89	90.88	91.08	**91.22**	91.17	90.99	91.94	92.19
Original Conv	90.58	90.58	90.73	90.78	91.08	90.68	91.78	91.80

Comparison of different loss functions. We first evaluate all the SphereConv operators with different loss functions. All the compared SphereConv operators use the 9-layer CNN architecture in the experiment. From the results in Table 3.1, one can observe that the SphereConv operators consistently outperforms the original convolutional operator. For the compared loss functions except A-Softmax and GA-Softmax, the effect on accuracy seems to less crucial than the SphereConv operators, but sigmoid W-Softmax is more flexible and thus works slightly better than the others. The sigmoid SphereConv operators with a suitably chosen parameter also works better than the others. Note that, W-Softmax loss is in fact comparable to the original softmax loss, because our SphereNet optimizes angles and the W-Softmax is derived from the original softmax loss. Therefore, it is fair to compare the SphereNet with W-Softmax and CNN with softmax loss. From Table 3.1, we can see SphereConv operators are consistently better than the convolutional operators. While we use a large-margin loss function like the A-Softmax [10] and the proposed GA-Softmax, the accuracy can be further boosted. One may notice that A-Softmax is actually cosine GA-Softmax. The superior performance of A-Softmax with SphereNet shows that our architecture is more suitable for the learning of angular loss. Moreover, our proposed large-margin loss (linear GA-Softmax) performs the best among all these compared loss

Table 3.2: Classification accuracy (%) with different network architectures.

SphereConv Operator	CNN-3	CNN-9	CNN-18	CNN-45	CNN-60	ResNet-32
Sigmoid (0.1)	82.08	91.13	91.43	89.34	87.67	90.94
Sigmoid (0.3)	81.92	**91.28**	91.55	89.73	87.85	**91.7**
Sigmoid (0.7)	82.4	91.18	**91.69**	89.85	88.42	91.19
Linear	**82.31**	91.15	91.24	**90.15**	**89.91**	91.25
Cosine	82.23	90.99	91.23	90.05	89.28	91.38
Original Conv	81.19	90.68	90.62	88.23	88.15	90.40

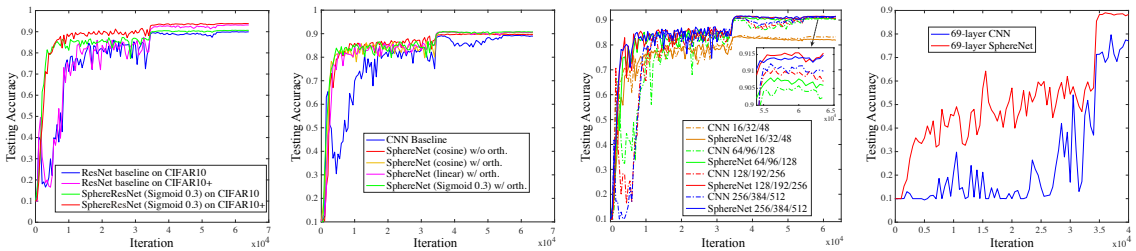
Table 3.3: Accuracy w/o ReLU.

SphereConv Operator	Acc. (%)
Sigmoid (0.1)	**86.29**
Sigmoid (0.3)	85.67
Sigmoid (0.7)	85.51
Linear	85.34
Cosine	85.25
CNN w/o ReLU	80.73

functions.

Comparison of different network architectures. We are also interested in how our SphereConv operators work in different architectures. We evaluate all the proposed SphereConv operators with the same architecture of different layers and a totally different architecture (ResNet). Our baseline CNN architecture follows the design of VGG network [42] only with different convolutional layers. For fair comparison, we use cosine W-Softmax for all SphereConv operators and original softmax for original convolution operators. From the results in Table 3.2, one can see that SphereNets greatly outperforms the CNN baselines, usually with more than 1% improvement. While applied to ResNet, our SphereConv operators also work better than the baseline. Note that, we use the similar ResNet architecture from the CIFAR-10 experiment in [2]. We do not use data augmentation for CIFAR-10 in this experiment, so the ResNet accuracy is much lower than the reported one in [2]. Our results on different network architectures show consistent and significant improvement over CNNs.

Comparison of different width (number of filters). We evaluate the SphereNet with different number of filters. Figure 3.3(c) shows the convergence of different width of



(a) ResNet vs. SphereResNet on CIFAR-10/10+ (b) CNN vs. SphereNet (orth.) on CIFAR-10 (c) Different width of SphereNet on CIFAR-10 (d) Deep CNN vs. SphereNet on CIFAR-10

Figure 3.3: Testing accuracy over iterations. (a) ResNet vs. SphereResNet. (b) Plain CNN vs. plain SphereNet. (c) Different width of SphereNet. (d) Ultra-deep plain CNN vs. ultra-deep plain SphereNet.

SphereNets. 16/32/48 means conv1.x, conv2.x and conv3.x have 16, 32 and 48 filters, respectively. One could observe that while the number of filters are small, SphereNet performs similarly to CNNs (slightly worse). However, while we increase the number of filters, the final accuracy will surpass the CNN baseline even faster and more stable convergence performance. With large width, we find that SphereNets perform consistently better than CNN baselines, showing that SphereNets can make better use of the width.

Learning without ReLU. We notice that SphereConv operators are no longer a matrix multiplication, so it is essentially a non-linear function. Because the SphereConv operators already introduce certain non-linearity to the network, we evaluate how much gain will such non-linearity bring. Therefore, we remove the ReLU activation and compare our SphereNet with the CNNs without ReLU. The results are given in Table 3.3. All the compared methods use 18-layer CNNs (with BatchNorm). Although removing ReLU greatly reduces the classification accuracy, our SphereNet still outperforms the CNN without ReLU by a significant margin, showing its rich non-linearity and representation power.

Convergence. One of the most significant advantages of SphereNet is its training stability and convergence speed. We evaluate the convergence with two different architectures: CNN-9 and ResNet-32. For fair comparison, we use the original softmax loss for all compared methods (including SphereNets). ADAM is used for the stochastic optimization and the learning rate is the same for all networks. From Figure 3.3(a), the SphereResNet converges significantly faster than the original ResNet baseline in both CIFAR-10 and

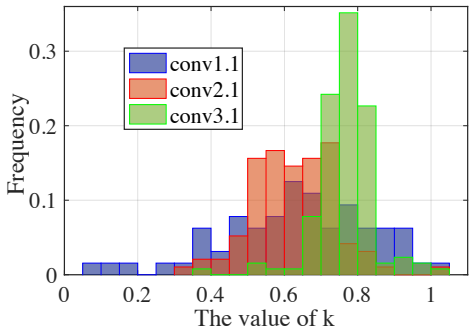


Figure 3.4: Frequency histogram of k.

CIFAR-10+ and the final accuracy are also higher than the baselines. In Figure 3.3(b), we evaluate the SphereNet with and without orthogonality constraints on kernel weights. With the same network architecture, SphereNet also converges much faster and performs better than the baselines. The orthogonality constraints also can bring performance gains in some cases. Generally from Figure 3.3, one could also observe that the SphereNet converges fast and very stably in every case while the CNN baseline fluctuates in a relative wide range.

Optimizing ultra-deep networks. Partially because of the alleviation of the covariate shift problem and the improvement of conditioning, our SphereNet is able to optimize ultra-deep neural networks without using residual units or any form of shortcuts. For SphereNets, we use the cosine SphereConv operator with the cosine W-Softmax loss. We directly optimize a very deep plain network with 69 stacked convolutional layers. From Figure 3.3(d), one can see that the convergence of SphereNet is much easier than the CNN baseline and the SphereNet is able to achieve nearly 90% final accuracy.

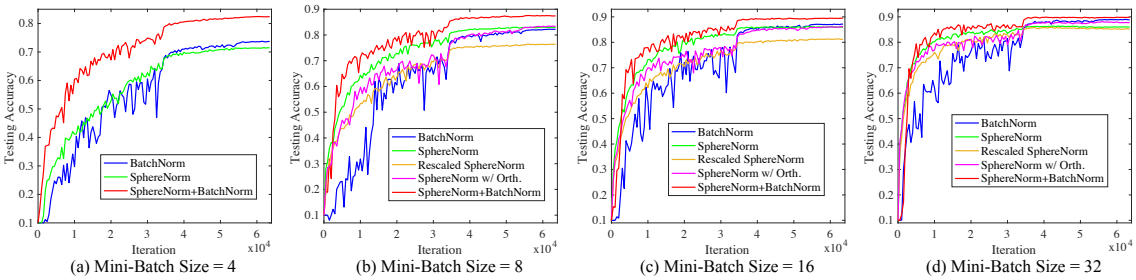
3.4.3 Preliminary Study towards Learnable SphereConv

Although the learnable SphereConv is not a main theme of this chapter, we still run some preliminary evaluations on it. For the proposed learnable sigmoid SphereConv, we learn the parameter k independently for each filter. It is also trivial to learn it in a layer-shared or network-shared fashsion. With the same 9-layer architecture used in Section subsection 3.4.2, the learnable SphereConv (with cosine W-Softmax loss) achieves 91.64%

on CIFAR-10 (without full data augmentation), while the best sigmoid SphereConv (with cosine W-Softmax loss) achieves 91.22%. In Figure 3.4, we also plot the frequency histogram of k in Conv1.1 (64 filters), Conv2.1 (96 filters) and Conv3.1 (128 filters) of the final learned SphereNet. From Figure 3.4, we observe that each layer learns different distribution of k. The first convolutional layer (Conv1.1) tends to uniformly distribute k into a large range of values from 0 to 1, potentially extracting information from all levels of angular similarity. The fourth convolutional layer (Conv2.1) tends to learn more concentrated distribution of k than Conv1.1, while the seventh convolutional layer (Conv3.1) learns highly concentrated distribution of k which is centered around 0.8. Note that, we initialize all k with a constant 0.5 and learn them with the back-prop.

3.4.4 Evaluation of SphereNorm

From subsection 3.4.2, we could clearly see the convergence advantage of SphereNets. In general, we can view the SphereConv as a normalization method (comparable to batch normalization) that can be applied to all kinds of networks. This section evaluates the challenging scenarios where the mini-batch size is small (results under 128 batch size could be found in subsection 3.4.2) and we use the same 9-layer CNN as in subsection 3.4.2. To be simple, we use the cosine SphereConv as SphereNorm. The softmax loss is used in both CNNs and SphereNets. From Figure 3.5, we could observe that SphereNorm achieves the final accuracy similar to BatchNorm, but SphereNorm converges faster and more stably. SphereNorm plus the orthogonal constraint helps convergence a little bit and rescaled SphereNorm does not seem to work well. While BatchNorm and SphereNorm are used together, we obtain the fastest convergence and the highest final accuracy, showing excellent compatibility of SphereNorm.



(a) Mini-Batch Size = 4 (b) Mini-Batch Size = 8 (c) Mini-Batch Size = 16 (d) Mini-Batch Size = 32

Figure 3.5: Convergence under different mini-batch size on CIFAR-10 dataset (We use the same experimental setting as subsection 3.4.2).

Table 3.4: Accuracy (%) on CIFAR-10+ & CIFAR-100.

Method	CIFAR-10+	CIFAR-100
ELU [86]	94.16	72.34
FitResNet (LSUV) [81]	93.45	65.72
ResNet-1001 [85]	**95.38**	**77.29**
Baseline ResNet-32 (softmax)	93.26	72.85
SphereResNet-32 (S-SW)	94.47	76.02
SphereResNet-32 (L-LW)	94.33	75.62
SphereResNet-32 (C-CW)	94.64	74.92
SphereResNet-32 (S-G)	**95.01**	**76.39**

3.4.5 Image Classification on CIFAR-10+ and CIFAR-100

We first evaluate the SphereNet in a classic image classification task. We use the CIFAR-10+ and CIFAR-100 datasets and perform random flip (both horizontal and vertical) and random crop as data augmentation (CIFAR-10 with full data augmentation is denoted as CIFAR-10+). We use the ResNet-32 as a baseline architecture. For the SphereNet of the same architecture, we evaluate sigmoid SphereConv operator ($k = 0.3$) with sigmoid W-Softmax ($k = 0.3$) loss (S-SW), linear SphereConv operator with linear W-Softmax loss (L-LW), cosine SphereConv operator with cosine W-Softmax loss (C-CW) and sigmoid SphereConv operator ($k = 0.3$) with GA-Softmax loss (S-G). In Table 3.4, we could see the SphereNet outperforms a lot of current state-of-the-art methods and is even comparable to the ResNet-1001 which is far deeper than ours. This experiment further validates our idea that learning on a hyperspheres constrains the parameter space to a more semantic and label-related one.

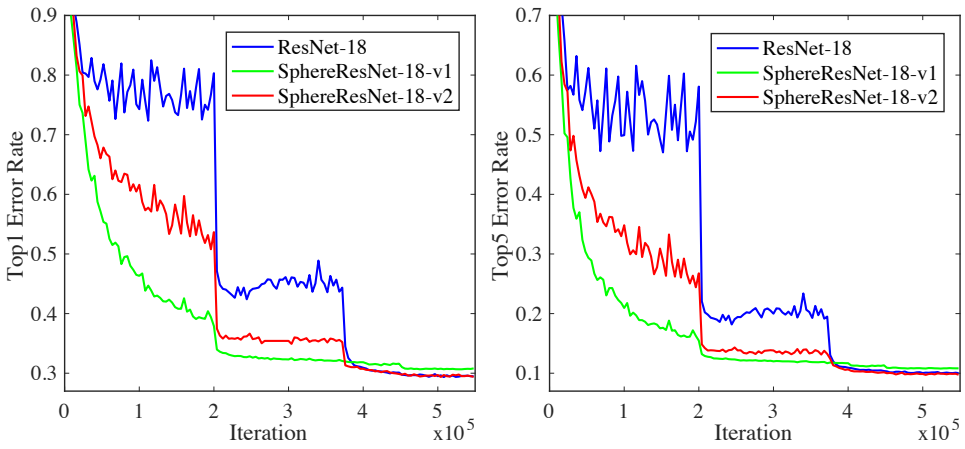


Figure 3.6: Validation error (%) on ImageNet.

3.4.6 Large-scale Image Classification on Imagenet-2012

We evaluate SphereNets on large-scale Imagenet-2012 dataset. We only use the minimum data augmentation strategy in the experiment (details are in section B.2). For the ResNet-18 baseline and SphereResNet-18, we use the same filter numbers in each layer. We develop two types of SphereResNet-18, termed as v1 and v2 respectively. In SphereResNet-18-v2, we do not use SphereConv in the 1×1 shortcut convolutions which are used to match the number of channels. In SphereResNet-18-v1, we use SphereConv in the 1×1 shortcut convolutions. Figure 3.6 shows the single crop validation error over iterations. One could observe that both SphereResNets converge much faster than the ResNet baseline, while SphereResNet-18-v1 converges the fastest but yields a slightly worse yet comparable accuracy. SphereResNet-18-v2 not only converges faster than ResNet-18, but it also shows slightly better accuracy.

CHAPTER 4

DECOUPLED NEURAL NETWORKS

4.1 Introduction

Convolutional neural networks have pushed the boundaries on a wide variety of vision tasks, including object recognition [42, 60, 61], object detection [87, 4, 88], semantic segmentation [3], etc. A significant portion of recent studies on CNNs focused on increasing network depth and representation ability via improved architectures such as shortcut connections [61, 89] and multi-branch convolution [60, 90]. Despite these advances, understanding how convolution naturally leads to discriminative representation and good generalization remains an interesting problem.

Current CNNs often encode the similarity between a patch $\boldsymbol{x}$ and a kernel $\boldsymbol{w}$ via inner product. The formulation of inner product $\langle \boldsymbol{w}, \boldsymbol{x} \rangle = \boldsymbol{w}^\top \boldsymbol{x}$ couples the semantic difference (*i.e.*, inter-class variation) and the intra-class variation in one unified measure. As a result, when the inner product between two samples is large, one can not tell whether the two samples have large semantic/label difference or have large intra-class variation. In order to better study the properties of CNN representation and further improve existing frameworks, we propose to explicitly decouple semantic difference and intra-class variation[1]. Specifically, we reparametrize the inner product with the norms and the angle, *i.e.*, $\|\boldsymbol{w}\|_2 \|\boldsymbol{x}\|_2 \cos(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. Our direct intuition comes from the the observation in Figure 4.1 where angle accounts for semantic/label difference and feature norm accounts for intra-class variation. The larger the feature norm, the more confident the prediction. Such naturally decoupled phenomenon inspires us to propose the decoupled convolution operators.

[1] Although the concepts of semantic difference and intra-class variation often refer to classification, they are extended to convolutions in this chapter. Specifically, semantic difference means the pattern similarity between local patch $\boldsymbol{x}$ and kernel $\boldsymbol{w}$, while intra-class variation refers to the energy of local patch $\boldsymbol{x}$ and kernel $\boldsymbol{w}$.

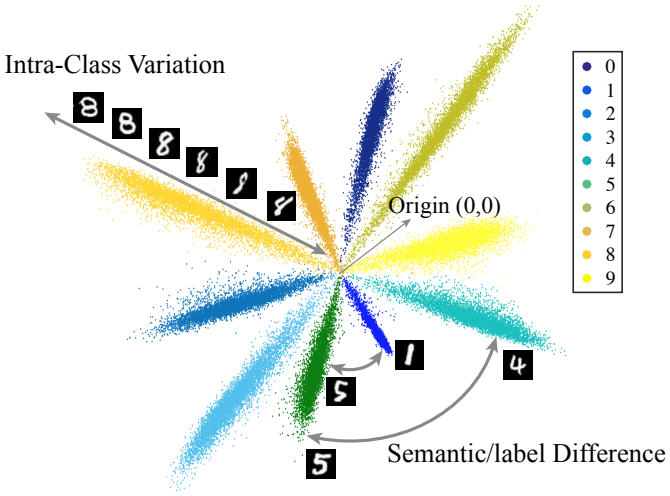


Figure 4.1: CNN learned features are naturally decoupled. These 2D features are output directly from the CNN by setting the feature dimension as 2.

We hope that decoupling norm and angle in inner product can better model the intra-class variation and the semantic difference in deep networks.

On top of the idea to decouple the norm and the angle in an inner product, we propose a novel decoupled network (DCNet) by generalizing traditional inner product-based convolution operators ($\|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos(\theta_{(\boldsymbol{w},\boldsymbol{x})})$) to decoupled operators. To this end, we define such operator as multiplication of a function of norms $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ and a function of angle $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. The decoupled operator provides a generic framework to better model the intra-class variation and the semantic difference, and the original CNNs are equivalent to setting $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ as $\|\boldsymbol{w}\|\|\boldsymbol{x}\|$ and $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ as $\cos(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. The magnitude function $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ models the intra-class variation while the angular function $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ models the semantic difference.

From the decoupling point of view, the original CNNs make a strong assumption that the intra-class variation can be linearly modeled via the multiplication of norms and the semantic difference is described by the cosine of the angle. However, this modeling approach is not necessarily optimal for all tasks. With the decoupled learning framework, we can either design the decoupled operators based on the task itself or learn them directly from data. The advantages of DCNets are in four aspects. First, DCNets not only allow us

to use some alternative functions to better model the intra-class variation and the semantic difference, but they also enable us to directly learn these functions rather than fixing them. Second, with bounded magnitude functions, DCNets can improve the problem conditioning as analyzed in [12], and therefore DCNets can converge faster while achieving comparable or even better accuracy than the original CNNs. Third, some instances of DCNets can have stronger robustness against adversarial attacks. We can squeeze the feature space of each class with a bounded $h(\cdot)$, which can bring certain robustness. Last, the decoupled operators are very flexible and architecture-agnostic. They could be easily adapted to any kind of architectures such as VGG [42], GoogleNet [60] and ResNet [61].

Specifically, we propose two different types of decoupled convolution operators: *bounded operators* and *unbounded operators*. We present multiple instances for each type of decoupled operators. Empirically, the bounded operators may yield faster convergence and better robustness against adversarial attacks, and the unbounded operators may have better representational power. These decoupled operators can also be either smooth or non-smooth, which can yield different behaviors. Moreover, we introduce a novel concept - *operator radius* for the decoupled operators. The operator radius describes the critical change of the derivative of the magnitude function $h(\cdot)$ with respect to the input $\|\boldsymbol{x}\|$. By jointly learning the operator radius via back-propagation, we further propose *learnable decoupled operators*. Moreover, we show some alternative ways to optimize these operators that improve upon standard back-propagation. Our contributions can be summarized as:

- Inspired by the observation that CNN-learned features are naturally decoupled, we propose an explicitly decoupled framework to study neural networks.
- We show that CNNs make a strong assumption to model the intra-class and inter-class variation, which may not be optimal. By decoupling the inner product, we are able to design more effective magnitude and angular functions rather than the original convolution for different tasks.
- In comparison to standard CNNs, DCNets have easier convergence, better accuracy and

stronger robustness.

4.2 Related Works

There are an increasing number of works [14, 16, 10, 9, 18, 11, 91, 92] that focus on improving the classification layer in order to increase the discriminativeness of learned features. [9] models the angular function for each class differently and defines a more difficult task than classification, improving the network generalization. Built upon [9], [10] further normalizes the weights of the last fully connected layer (*i.e.*, classification layer) and reported improved results on face recognition. [14, 16, 11] normalize the input features before entering the last fully connected layer, achieving promising performance on face recognition. However, these existing works can be viewed as heuristic modifications and are often restricted to the last fully connected layer. In contrast, the decoupled learning provides a more general and systematic way to study the CNNs. In our framework, the previous work can be viewed as proposing a new magnitude function $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ or angular function $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ for the last fully connected layer. For example, normalizing the weights is to let $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ be $\|\boldsymbol{x}\|$ and normalizing the input is equivalent to $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \|\boldsymbol{w}\|$.

[12] proposes a deep hyperspherical learning framework which directly makes $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ equal to 1 such that all the activation outputs only depend on $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. The framework provides faster convergence compared to the original CNNs, but is somehow restricted in the sense that $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ is only allowed to be 1, and therefore can be sub-optimal in some cases. From the decoupling perspective, hyperspherical learning only cares about the semantic difference and aims to compress the intra-class variation to a space that is as small as possible, while the decoupled framework focuses on both. As a non-trivial generalization of [12], our decoupled network is a more generic and unified framework to model both intra-class variation and semantic difference, providing the flexibility to design or learn both magnitude function $h(\cdot)$ and angular function $g(\cdot)$.

4.3 Decoupled Networks

4.3.1 Reparametrizing Convolution via Decoupling

For a conventional convolution operator $f(\cdot,\cdot)$, the output is calculated by the inner product of the input patch $\boldsymbol{x}$ and the filter $\boldsymbol{w}$ (both $\boldsymbol{x}$ and $\boldsymbol{w}$ are vectorized into columns):

$$f(\boldsymbol{w},\boldsymbol{x}) = \langle \boldsymbol{w},\boldsymbol{x}\rangle = \boldsymbol{w}^\top \boldsymbol{x}, \tag{4.1}$$

which can be further formulated as a decoupled form that separates the norm and the angle:

$$f(\boldsymbol{w},\boldsymbol{x}) = \|\boldsymbol{w}\|\,\|\boldsymbol{x}\|\,cos(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{4.2}$$

where $\theta_{(\boldsymbol{w},\boldsymbol{x})}$ is the angle between $\boldsymbol{x}$ and $\boldsymbol{w}$. Our proposed decoupled convolution operator takes the general form of

$$f_d(\boldsymbol{w},\boldsymbol{x}) = h(\|\boldsymbol{w}\|,\|\boldsymbol{x}\|)\cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{4.3}$$

which explicitly decouples the norm of $\boldsymbol{w},\boldsymbol{x}$ and the angle between them. We define $h(\|\boldsymbol{w}\|,\|\boldsymbol{x}\|)$ as the magnitude function and $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ as the angular activation function. It is easy to see that the decoupled convolution operator includes the original convolution operator as a special case. As illustrated in Figure 4.1, the semantic difference and intra-class variation are usually decoupled and very suitable for this formulation. Based on the decoupled operator, we propose several alternative ways to model the semantic difference and intra-class variation.

4.3.2 Decoupled Convolution Operators

We discuss how to better model the intra-class variation, and then give a few instances of the decoupled operator.

On Better Modeling of the Intra-class Variation

Hyperspherical learning [12] has discussed the modeling of the inter-class variation (*i.e.*, the angular function). The design of angular function $g(\cdot)$ is relatively easy but restricted, because it only takes the angle as input. In contrast, the magnitude function $h(\cdot)$ takes the norm of $\boldsymbol{w}$ and the norm of $\boldsymbol{x}$ as two inputs, and therefore it is more complicated to design. $\|\boldsymbol{w}\|$ is the intrinsic property of a kernel itself, corresponding to the importance of the kernel rather than the intra-class variation of the inputs. Therefore, we tend not to include $\|\boldsymbol{w}\|$ into the magnitude function $h(\cdot)$. Moreover, removing $\|\boldsymbol{w}\|$ from $h(\cdot)$ indicates that all kernels (or operators) are assigned with equal importance, which encourages the network to make decision based on as many kernels as possible and therefore may make the network generalize better. However, incorporating the kernel importance to the network learning can improve the representational power and may be useful when dealing with a large-scale dataset with numerous categories. By combining $\|\boldsymbol{w}\|$ back to $h(\cdot)$, the operators become *weighted decoupled operators*. There are multiple ways of incorporating $\|\boldsymbol{w}\|$ back to the magnitude function. We will discuss and empirically evaluate these variants later.

Bounded Decoupled Operators

The output of the bounded operators must be bounded by a finite constant regardless of its input and kernel, namely $|f_d(\boldsymbol{w}, \boldsymbol{x})| \leq c$ where c is a positive constant. For simplicity, we first consider the decoupled operator without the norm of the weights (*i.e.*, $\|\boldsymbol{w}\|$ is not included in $h(\cdot)$).

Hyperspherical Convolution. If we let $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \alpha$, we will have the hyperspherical convolution (SphereConv) with the following decoupled form:

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{4.4}$$

where $\alpha > 0$ controls the output scale. $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ depends on the geodesic distance on

the unit hypersphere and typically outputs value from -1 to 1, so the final output is in $[-\alpha, \alpha]$. Usually, we can use $\alpha = 1$, which reduces to SphereConv [12] in this case. Geometrically, SphereConv can be viewed as projecting $\boldsymbol{w}$ and $\boldsymbol{x}$ to a hypersphere and then performing inner product (if $g(\theta) = \cos(\theta)$). Based on [12], SphereConv improves the problem conditioning in neural networks, making the network converge better.

Hyperball Convolution. The hyperball convolution (BallConv) uses $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \alpha \min(\|\boldsymbol{x}\|, \rho)/\rho$ as its magnitude function. The specific form of the BallConv is

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \cdot \frac{\min(\|\boldsymbol{x}\|, \rho)}{\rho} \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{4.5}$$

where ρ controls the saturation threshold for the input norm $\|\boldsymbol{x}\|$ and α scales the output range. When $\|\boldsymbol{x}\|$ is larger than ρ, then the magnitude function will be saturate and output α. When $\|\boldsymbol{x}\|$ is smaller than ρ, the magnitude function grows linearly with $\|\boldsymbol{x}\|$. Geometrically, BallConv can be viewed as projecting $\boldsymbol{w}$ to a hypersphere and projecting the input $\boldsymbol{x}$ to a hyperball, and then performing the inner product (if $g(\theta) = \cos(\theta)$). Intuitively, BallConv is more robust and flexible than SphereConv in the sense that SphereConv may amplify the $\boldsymbol{x}$ with very small $\|\boldsymbol{x}\|$, because $\boldsymbol{x}$ with small $\|\boldsymbol{x}\|$ and the same direction as $\boldsymbol{w}$ could still produce the maximum output. It makes SphereConv sensitive to perturbations to $\boldsymbol{x}$ with small norm. In contrast, BallConv will not have such a problem, because the multiplicative factor $\|\boldsymbol{x}\|$ can help to decrease the output if $\|\boldsymbol{x}\|$ is small. Moreover, small $\|\boldsymbol{x}\|$ indicates that the local patch is not informative and should not be emphasized. In this sense, BallConv is better than SphereConv. In terms of convergence, the BallConv can still help the network convergence because its output is bounded with the same range as SphereConv.

Hyperbolic Tangent Convolution. We present a smooth decoupled operator with bounded output called hyperbolic tangent convolution (TanhConv). The TanhConv uses a hyperbolic tangent function to replace the step function in the BallConv and can be formulated as

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \tanh\left(\frac{\|\boldsymbol{x}\|}{\rho}\right) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{4.6}$$

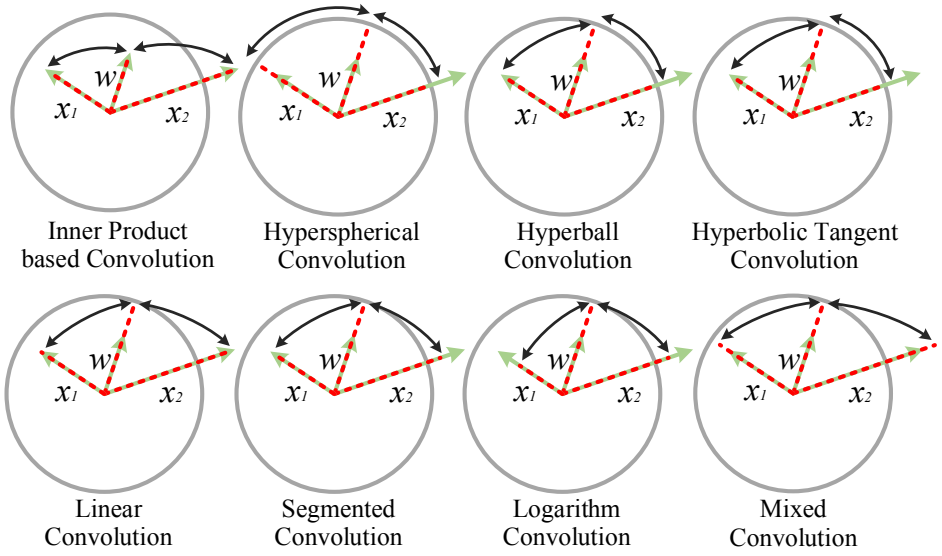


Figure 4.2: Geometric interpretations for decoupled convolution operators. Green denotes the original vectors, and red denotes the projected vectors.

where $\tanh(\cdot)$ denotes the hyperbolic tangent function and ρ is parameter controlling the decay curve. The TanhConv can be viewed as a smooth version of BallConv, which not only shares the same advantages as BallConv but also has more convergence gain due to its smoothness [86].

Unbounded Decoupled Operators

Linear Convolution. One of the simplest unbounded decoupled operators is the linear convolution (LinearConv):

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \|\boldsymbol{x}\| \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{4.7}$$

where α controls the output scale. LinearConv differs the original convolution in the sense that it projects the weights to a hypersphere and has a parameter to control the slope.

Segmented Convolution. We propose a segmented convolution (SegConv) which takes the following form:

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \begin{cases} \alpha \|\boldsymbol{x}\| \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), & 0 \leq \|\boldsymbol{x}\| \leq \rho \\ (\beta \|\boldsymbol{x}\| + \alpha\rho - \beta\rho) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), & \rho < \|\boldsymbol{x}\| \end{cases}, \tag{4.8}$$

where α controls the slope when $\|\boldsymbol{x}\| \leq \rho$ and β controls the slope when $\|\boldsymbol{x}\| > \rho$. ρ is the change point of the gradient of the magnitude function w.r.t. $\|\boldsymbol{x}\|$. SegConv is a flexible multi-range linear function corresponding to $\|\boldsymbol{x}\|$. Both LinearConv and BallConv are special cases of SegConv.

Logarithm Convolution. We present another smooth decoupled operator with unbounded output, logarithm convolution (LogConv). LogConv uses a Logarithm function for the norm of the input $\|\boldsymbol{x}\|$ and can be formulated as

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \log(1 + \|\boldsymbol{x}\|) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{4.9}$$

where α controls the base of logarithm and is used to adjust the curvature of the logarithm function.

Mixed Convolution. Mixed convolution (MixConv) combines multiple decoupled convolution operators and enjoys better flexibility. Because the mixed convolution has many possible combinations, we only consider the additive combination of LinearConv and LogConv as an example:

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \big(\alpha \|\boldsymbol{x}\| + \beta \log(1 + \|\boldsymbol{x}\|)\big) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{4.10}$$

which combines LogConv and LinearConv, becoming more flexible than both original operators.

Properties of Decoupled Operators

Operator Radius. Operator radius is defined to describe the gradient change point of the magnitude function. Operator radius differentiates two stages of the magnitude function. The two stages usually have different gradient ranges and therefore behave differently during optimization. We let ρ denote the operator radius in each decoupled operator. For BallConv, when $\|\boldsymbol{x}\|$ is smaller than ρ, the magnitude function will be activated and it will grow with $\|\boldsymbol{x}\|$ linearly. When $\|\boldsymbol{x}\|$ is larger than ρ, then the magnitude function will be

deactivated and output a constant. For SegConv, $\|\boldsymbol{x}\| = \rho$ is the change point of the magnitude function's slope. The operator radius of some decoupled operators (SphereConv, LinearConv, LogConv) is defined to be zero, indicating that they have no operator radius. The decoupled operator with non-zero operator radius is similar to a gated operator where $\|\boldsymbol{x}\| = \rho$ serves as the switch.

Boundedness. The Boundedness of a decoupled operator may affect its convergence speed and robustness. [12] shows that using a bounded operator can improve the convergence due to two reasons. First, bounded operators lead to better problem conditioning in training a deep network via stochastic gradient descent. Second, bounded operators make the variance of the output small and partially address the internal covariate shift problem. The bounded operators can also constrain the Lipschitz constant of a neural network, making the entire network more smooth. The Lipschitz constant of a neural network is shown to be closely related to its robustness against adversarial perturbation [93]. In contrast, the unbounded operators may have stronger approximation power and flexibility than the bounded ones.

Smoothness. The smoothness of the magnitude function is closely related to the approximation ability and the convergence behavior. In general, using a smooth magnitude function could have better approximation rate [94] and may also lead to more stable and faster convergence [86]. However, a smooth magnitude function may also be more computationally expensive, since it could be more difficult to approximate a smooth function with polynomials.

4.3.3 Geometric Interpretations

All the decoupled convolution operators have very clear geometric interpretations, as illustrated in Figure 4.2. Because all decoupled operators normalize the kernel weights, all the weights are already on the unit hypersphere. SphereConv also projects the input vector $\boldsymbol{x}$ on the unit hypersphere and then computes the similarity between $\boldsymbol{w}$ and $\boldsymbol{x}$ based on

the geodesic distance on the hypersphere (multiplied by a scaling factor α). Therefore, its output is bounded from $-\alpha$ to α and only depends on the directions of $\boldsymbol{w}$ and $\boldsymbol{x}$ (suppose $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ is in the range of $[-1, 1]$).

BallConv first projects the input vector $\boldsymbol{x}$ to a hyperball and then computes the similarity based on the projected $\boldsymbol{x}$ inside the hyperball and the normalized $\boldsymbol{w}$ on surface of the hyperball. Specifically, BallConv projects $\boldsymbol{x}$ to the hypersphere if $\|\boldsymbol{x}\| > \rho$. TanhConv is a smoothed BallConv and has similar geometric interpretation, but TanhConv is differentible everywhere and has soft boundary around the operator radius $\|\boldsymbol{x}\| = \rho$. TanhConv can be viewed as performing projection to a soft hyperball.

SegConv is more flexible than both SphereConv and BallConv. By using certain parameters, SegConv can reduce to either SphereConv or BallConv. SegConv essentially adjusts the norm of the input $\boldsymbol{x}$ with a multi-range linear multiplicative factor. Geometrically, such a factor will either push the vector close to the hypersphere or away from the hypersphere depending on the selection of α and β. For example, we consider the case where $\alpha = 1$ and $0 < \beta < 1$. When $\|\boldsymbol{x}\| \leq \rho$, the magnitude function $h(\cdot)$ in SegConv will directly output $\|\boldsymbol{x}\|$. When $\|\boldsymbol{x}\| > \rho$, $h(\cdot)$ in SegConv will output a value smaller than $\|\boldsymbol{x}\|$, as shown in Figure 4.2.

LinearConv is the simplest unbounded operator and its magnitude function grows linearly with $\|\boldsymbol{x}\|$. When $\alpha = 1$, the magnitude function $h(\cdot)$ in LinearConv simply outputs $\|\boldsymbol{x}\|$, which does not perform any projection.

LogConv use a logarithm function to transform the norm of the input $\boldsymbol{x}$. After such nonlinear transformation on $\boldsymbol{x}$, LogConv computes similarity based on the transformed input $\boldsymbol{x}$ and the normalized weights on a hypersphere.

4.3.4 Design of the Angular Activation Function

The design of the angular function $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ mostly follows the deep hyperspherical learning [12]. We use four different types of $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ in this chapter. The linear angular

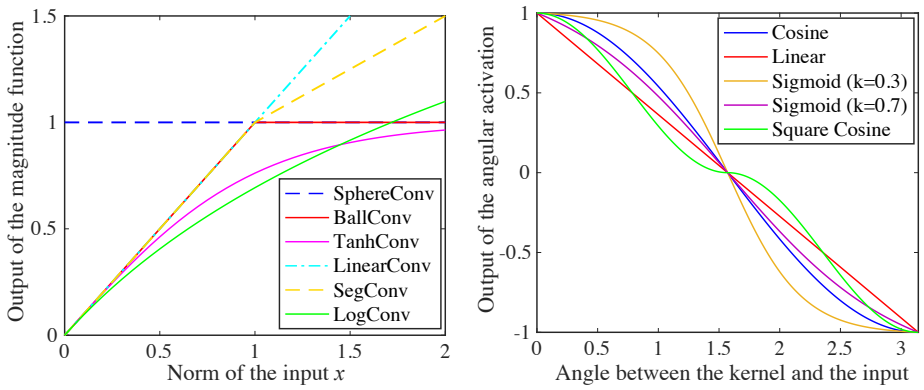


Figure 4.3: Magnitude function ($\rho = 1$) and angular activation function.

activation is defined as

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = -\frac{2}{\pi}\theta_{(\boldsymbol{w},\boldsymbol{x})} + 1, \tag{4.11}$$

whose output grows linearly with the angle $\theta_{(\boldsymbol{w},\boldsymbol{x})}$. The cosine angular activation is defined as

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = \cos(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{4.12}$$

which is also used by the original convolution operator. Moreover, the sigmoid angular activation is defined as

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = \frac{1+\exp(-\frac{\pi}{2k})}{1-\exp(-\frac{\pi}{2k})} \cdot \frac{1-\exp(\frac{\theta_{(\boldsymbol{w},\boldsymbol{x})}}{k}-\frac{\pi}{2k})}{1+\exp(\frac{\theta_{(\boldsymbol{w},\boldsymbol{x})}}{k}-\frac{\pi}{2k})}, \tag{4.13}$$

where k controls the curvature. Additionally, we also propose a square cosine angular activation function:

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = \text{sign}(\cos(\theta)) \cdot \cos^2(\theta), \tag{4.14}$$

which can encourage a degree of angular margin near the decision boundary and may improve network generalization. In addition to fixing these angular activations prior to training, we can also jointly learn the parameter k in the sigmoid activation using back-propagation, which is a learnable angular activation [12]. Figure 4.3 shows the curves of these angular activation functions.

4.3.5 Weighted Decoupled Operators

All the decoupled operators we have discussed normalize the kernel weights $\boldsymbol{w}$ and the magnitude functions do not take the weights into consideration. Although empirically we find that the standard decoupled operators work better than the weighted ones in most cases, we still consider weighted decoupled operators, which incorporate $\|\boldsymbol{w}\|$ into the magnitude function, in order to improve the operator's flexibility. We propose two straightforward ways to combine $\|\boldsymbol{w}\|$: linear and nonlinear.

Linearly Weighted Decoupled Operator. Similar to the original inner produce-based convolution, we can directly multiply the norm of weights into the magnitude function, which makes the decoupled operators linearly weighted. For example, SphereConv will become $f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha\|\boldsymbol{w}\| \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. Notably, linearly weighted LinearConv will become the original inner produce-based convolution.

Nonlinearly Weighted Decoupled Operator. Compared to linearly weighted decoupled operators, the norm of the weights are incorporated into the magnitude function in a nonlinear way. Taking TanhConv as an example, we could formulate the nonlinearly weighted TanhConv as

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \tanh(\frac{1}{\rho}\|\boldsymbol{x}\| \cdot \|\boldsymbol{w}\|) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}). \tag{4.15}$$

We can also formulate the nonlinearly weighted TanhConv in an alternative way:

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \tanh(\frac{1}{\rho}\|\boldsymbol{w}\|) \cdot \tanh(\frac{1}{\rho}\|\boldsymbol{x}\|) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}). \tag{4.16}$$

The first nonlinearly weighted formulation couples $\|\boldsymbol{x}\|$ and $\|\boldsymbol{w}\|$ by multiplication and then perform a nonlinear transformation, while the second one performs nonlinear transformations separately for $\|\boldsymbol{x}\|$ and $\|\boldsymbol{w}\|$, and then multiplies them. In practice, the linearly weighted operators are favored over nonlinearly weighted ones due to the simplicity.

4.3.6 Learnable Decoupled Operators

Because our decoupled operators usually have hyperparameters, we usually need to do cross-validation in order to choose suitable parameters, which is time-consuming and sub-optimal. To address this, we can learn these parameters jointly with network weight training via back-propagation. We propose learnable decoupled operators which perform hyperparameter learning with $h(\cdot)$ and $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. For example, [12] proposed to learn the hyperparameters of sigmoid angular function. By making both $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ and $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ learnable, we can greatly enhance the representational power and flexibility.

However, making the decoupled operators too flexible (*i.e.*, too many learnable parameters) may require a prohibitive amount of training data to achieve good generalization. In order to achieve an effective trade-off, we only investigate learning the operator radius ρ via back-propagation during the network training.

4.4 Improving the Optimization for DCNets

We propose several tricks to improve the optimization of DCNets and enable DCNets to converge to a better local minima. More analysis and discussion of weight projection and weighted gradients are provided in section C.7.

4.4.1 Weight Projection

The forward pass of DCNets is not dependent on the norm of the weights $\|\boldsymbol{w}\|$, because the decoupled operators take the normalized weights as input. However, $\|\boldsymbol{w}\|$ will significantly affect the backward pass. Taking SphereConv as an example, we compute the gradient w.r.t. $\boldsymbol{w}$:

$$\frac{\partial}{\partial \boldsymbol{w}}\left(\hat{\boldsymbol{w}}^{\top}\hat{\boldsymbol{x}}\right) = \frac{\hat{\boldsymbol{x}} - \hat{\boldsymbol{w}}^{\top}\hat{\boldsymbol{x}} \cdot \hat{\boldsymbol{w}}}{\|\boldsymbol{w}\|}, \tag{4.17}$$

where $\hat{\boldsymbol{w}} = \boldsymbol{w}/\|\boldsymbol{w}\|$ and $\hat{\boldsymbol{x}} = \boldsymbol{x}/\|\boldsymbol{x}\|$. In comparison, $\|\boldsymbol{w}\|$ will not affect the gradient w.r.t $\boldsymbol{w}$ in inner product. From (Equation 4.17), large $\|\boldsymbol{w}\|$ can make the gradients very small

so that the backward pass is not able to update the weights effectively. To address this issue, we propose *weight projection* to control the norm of the weights. Weight projection performs $\boldsymbol{w} \leftarrow s \cdot \hat{\boldsymbol{w}}$ every certain number of iterations where $\leftarrow$ denotes the replacement operation. s is a positive constant which controls the norm of the gradient (we use $s = 1$ in our experiments). In general, larger s leads to smaller gradients. Note that, weight projection cannot be used in the weighted decoupled operators, because $\|\boldsymbol{w}\|$ will affect the forward pass. We can only apply weight projection to our standard decoupled operators.

4.4.2 Weighted Gradients

From (Equation 4.17), we observe that we could simply multiply $\|\boldsymbol{w}\|$ to (Equation 4.17) to eliminate the effect of $\|\boldsymbol{w}\|$ on the backward pass. We update the weights with the following:

$$\Delta \boldsymbol{w} = \|\boldsymbol{w}\| \cdot \frac{\partial}{\partial \boldsymbol{w}} \left(\hat{\boldsymbol{w}}^\top \hat{\boldsymbol{x}}\right) = \hat{\boldsymbol{x}} - \hat{\boldsymbol{w}}^\top \hat{\boldsymbol{x}} \cdot \hat{\boldsymbol{w}}, \tag{4.18}$$

which does not depend on $\|\boldsymbol{w}\|$ and is called *weighted gradients*. Using the proposed weighted gradients for back-propagation, we can also prevent the gradients from being affected by the norm of the weights.

4.4.3 Pretraining as a Better Initialization

We find that DCNets may sometimes be trapped into a bad local minima and yield a less competitive accuracy while trained on large-scale datasets (*e.g.*, ImageNet). Because the decoupled operators have stronger nonlinearity, its loss landscape may be more complex than the original convolution. The most straightforward way to improve the optimization is to use a better initialization. To this end, we use a CNN model that has the same structure and is pretrained on the same training set to initialize the DCNet.

4.5 Discussions

Why Decoupling? Decoupling the intra-class and inter-class variation gives us the flexibility to design better models that are more suitable for a given task. Inner product-based convolution is computationally attractive but not necessarily optimal. The original convolution makes an assumption that the intra-class and inter-class variations are modeled by $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \|\boldsymbol{w}\|\|\boldsymbol{x}\|$ and $g(\theta) = \cos(\theta)$, respectively. Such assumptions may not be optimal. $h(\cdot)$ and $g(\cdot)$ can be task-driven in our novel decoupled framework.

Flexibility of Decoupled Operators. There are numerous design options for the magnitude and angular function. The original convolution can be viewed as a special decoupled operator. Moreover, we can parametrize a decoupled operator with a few learnable parameters and learn them via back-propagation. However, there is a delicate tradeoff between the size of the training data, the generalization of the network and the flexibility of the decoupled operator. Generally, given a large enough dataset, the network generalization improves with more learnable parameters.

A Unified Learning Framework for CNNs. The decoupled formulation provides a unified learning framework for CNNs. Consider a standard CNN with ReLU, we write the convolution and ReLU as $\max(0, \|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos(\theta))$ which can be written as $\|\boldsymbol{w}\|\|\boldsymbol{x}\| \cdot \max(0, \cos(\theta))$. Such formulation can be viewed as a decoupled operator where $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \|\boldsymbol{w}\|\|\boldsymbol{x}\|$ and $g(\theta) = \max(0, \cos(\theta))$. We can jointly consider the convolution operator and nonlinear activation in the decoupled framework. It is possible to learn one single function $g(\cdot)$ that represents both angular activation and the nonlinearity, which is why the square cosine angular activation works well without ReLU.

Network Regularization. In most instances of DCNets, the ℓ_2 weight decay is no longer suitable. [12] uses an orthonormal constraint $\|\boldsymbol{W}^\top\boldsymbol{W} - \boldsymbol{I}\|_F^2$ to regularize the network, where $\boldsymbol{W}$ is the weight matrix whose columns are the kernel weights and $\boldsymbol{I}$ is identity matrix. We also propose an orthogonal constraint $\|\boldsymbol{W}^\top\boldsymbol{W} - \mathrm{diag}(\boldsymbol{W}^\top\boldsymbol{W})\|_F^2$.

Network Architecture. Due to the non-linear nature of DCNet, the performance of specific $h(\cdot)$ and $g(\cdot)$ is dependent on the choice of architecture. An interesting challenge for future work is to inverstigate the link between the architecture and the choise of $h(\cdot)$ and $g(\cdot)$.

4.6 Experiments and Results

General. We evaluate both accuracy and robustness of DCNets on objection recognition. For all decoupled operators, we use the standard softmax loss if not otherwise specified.

Training. The architecture for each task and the training details are given in section C.1. For CIFAR, the network is trained by ADAM with 128 batch size. The learning rate starts from 0.001. For ImageNet, we use SGD with momentum 0.9 and batch size 40. The learning rate starts from 0.1. For adversarial attacks, the networks are trained by ADAM. All learning rates are divided by 10 when the error plateaus.

Implementation Details. For all decoupled operators that have non-zero operator radius (*i.e.*, $\rho \neq 0$), we will learn the operator radius from the training data via back-propagation. More details are provided in section C.2.

4.6.1 Object Recognition

CIFAR-10 and CIFAR-100

Weighted Decoupled Operators. We first compare the weighted decoupled operators and the standard ones. Because the weights are incorporated into the forward pass in the weighted decoupled operators, the optimization tricks like weight projection and weighted gradients are not applicable. Therefore, the weighted operators simply use the conventional gradients to perform back-propagation. For standard decoupled operators, we show the results using standard optimization, weight projection and weighted gradients. From the results of TanhConv in Table 4.1, weighted decoupled operators do not show obvious advantages.

Optimization Tricks. We propose weight projection and weighted gradients to facilitate

Table 4.1: Evaluation of weighted operators (TanhConv) on CIFAR-100.

Method	Error
Linearly Weighted Decoupled Operator	22.95
Nonlinearly Weighted Decoupled Operator ((Equation 4.15))	23.03
Nonlinearly Weighted Decoupled Operator ((Equation 4.16))	23.38
Decoupled Operator (Standard Gradients)	23.09
Decoupled Operator (Weight Projection)	**21.17**
Decoupled Operator (Weighted Gradients)	21.45

Table 4.2: Testing error (%) of plain CNN-9 without BN on CIFAR-100. "N/C" indicates that the model can not converge. "-" denotes no result. The results of different columns belong to different angular activation.

Method	Linear	Cosine	Square Cosine
CNN Baseline	-	35.30	-
LinearConv	33.39	31.76	N/C
TanhConv	**32.88**	31.88	**34.26**
SegConv	34.69	**30.34**	N/C

the optimization of DCNets. These two tricks essentially amplify the original gradient and make the backward update more effective. From Table 4.1, we observe that both weight projection and weighted gradients work much better than the competing methods.

Learning without Batch Normalization. Batch Normalization (BN) [44] is usually crucial for training a well-performing CNN, but the results in Table 4.2 show that our decoupled operators can perform much better than the original convolution even without BN.

Learning without ReLU. Our decoupled operators naturally have strong nonlinearity, because our decoupled convolution is no longer a linear matrix multiplication. In Table 4.3, square cosine angular activation works extremely well in plain CNN-9, even better than the networks with ReLU. The results show that using suitable $h(\cdot)$ and $g(\cdot)$ can lead to significantly better accuracy than the baseline CNN with ReLU, even if our DCNet does not use ReLU at all.

Comparison among Different Decoupled Operators. We compare different decoupled operators on both plain CNN-9 and ResNet-32. All the compared decoupled operators are unweighted and use weight projection during optimization. The standard softmax loss and BN are used in all networks. For plain CNN-9, we compare the case with and without

Table 4.3: Testing error rate (%) of plain CNN-9 on CIFAR-100. Note that, BN is used in all compared models. Baseline is the original plain CNN-9.

Method	Cosine w/o ReLU	Square Cosine w/o ReLU	Cosine w/ ReLU	Square Cosine w/ ReLU
Baseline	58.24	-	26.01	-
SphereConv	33.31	25.90	26.00	26.97
BallConv	**31.81**	25.43	25.18	26.48
TanhConv	32.27	25.27	25.15	26.94
LinearConv	36.49	24.36	**24.81**	25.14
SegConv	33.57	**24.29**	24.96	**25.04**
LogConv	33.62	24.91	25.17	25.85
MixConv	33.46	24.93	25.27	25.77

Table 4.4: Testing error rate (%) of ResNet-32 on CIFAR-100.

Method	Linear	Cosine	Square Cosine
ResNet Baseline	-	26.69	-
SphereConv	21.79	21.44	24.40
BallConv	21.44	21.12	24.31
TanhConv	21.6	21.17	24.77
LinearConv	21.09	22.17	21.31
SegConv	**20.86**	**20.91**	**20.88**
LogConv	21.84	21.08	22.86
MixConv	21.02	21.28	21.81

ReLU. The results in Table 4.3 show that DCNets significantly outperform the baseline. In particular, our DCNet with SegConv and square cosine can achieve 24.29% even without ReLU, which is even better than the networks with ReLU. For ResNet-32, our DCNets also consistently outperform the baseline with a considerable margin. The results further verify that the intra-class and inter-class variation assumptions of the original CNN are not optimal.

Convergence. We also evaluate the convergence of DCNets using the architecture of ResNet-32. The convergence curves in Figure 4.4 show that the decoupled operators are able to converge and generalize better than original convolution operators on CIFAR-100 dataset.

Comparison to the state-of-the-art. Table 4.5 shows that our DCNet-32 has very competitive accuracy compared to ResNet-1001. In order to achieve best accuracy, we use the weight-normalized softmax loss [12]. We also find that using SGD further improves the

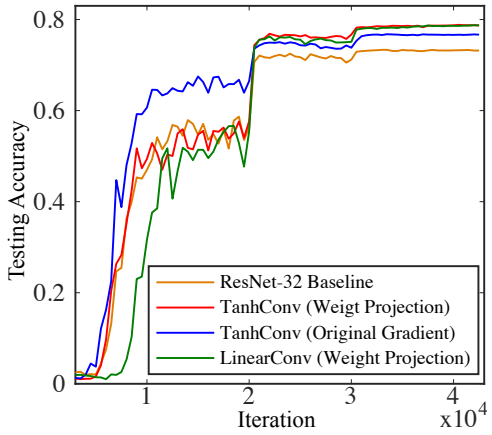


Figure 4.4: Convergence of DCNets on CIFAR-100.

Table 4.5: Comparison to the state-of-the-art on CIFAR-10 and CIFAR-100.

Method	CIFAR-10	CIFAR-100
ResNet-110-original [61]	6.61	25.16
ResNet-1001 [85]	4.92	**22.71**
ResNet-1001 (64 mini-batch size) [85]	**4.64**	-
DCNet-32 (TanhConv + Cosine)	**4.75**	21.12
DCNet-32 (LinearConv + Square Cos.)	5.34	**20.23**

accuracy of DCNets. Experiments on SGD-trained models are provided in section C.6.

ImageNet-2012

Standard ResNet. We first evaluate the DCNets with the standard ResNet-18. All presented decoupled operators use the cosine angular activation. [12] shows that SphereConv can perform comparably to the baseline on ImageNet only when the network is wide enough. Using the weight projection and pretrained model initialization, SphereConv is comparable to the baseline even on narrow networks. Most importantly, TanhConv and LinearConv achieve better accuracy than the baseline ResNet. The learned filters of DCNets on ImageNet are also provided in section C.5.

Modified ResNet. We also evaluate decoupled operators with a modified ResNet, similar to SphereFace networks [10], to better show the advantages of decoupled operators. DCNets can be trained from scratch and outperform the baseline by 1%. Moreover, DCNets can converge stably in very challenging scenarios. From Table 4.6, we observe that DCNets

Table 4.6: Center-crop Top-5 error (%) of standard ResNet-18 and modified ResNet-18 on ImageNet-2012. * indicates we use the pretrained model of original CNN on ImageNet-2012 as initialization (see subsection 4.4.3).

Method	Standard ResNet-18 w/ BN	Modified ResNet-18 w/ BN	Modified ResNet-18 w/o BN
Baseline	12.63	12.10	N/C
SphereConv	12.68*	11.55	13.30
LinearConv	**11.99***	11.50	N/C
TanhConv	12.47*	**11.10**	**12.79**

can converge to a decent accuracy without BN, while the baseline model fails to converge without BN.

4.6.2 Robustness against Adversarial attacks

We evaluate the robustness of DCNets. DCNets in this subsection use standard gradients and are trained without any optimization trick. Fast gradient sign method (FGSM) [95] and basic iterative method (BIM) [96] are used to attack the networks. Experimental details and more experiments are given in section C.1 and section C.3, respectively.

White-box Adversarial Attacks

We run white-box attacks on both naturally trained models and FGSM-trained models on CIFAR-10 (results shown in Table 4.7). "None" attacks mean that all the testing samples are normal. For naturally trained models, all DCNet variants show significantly better robustness over the baseline, with naturally trained TanhConv being most resistant. With adversarial training, while DCNets achieve the best robustness, SphereConv is particularly resistant against BIM attack. We speculate that the tight spherical constraint strongly twists the data manifold so that the adversarial gradient updates can only result in small gains.

Table 4.7: White-box attacks on CIFAR-10. Performance is measured in accuracy (%). The first three rows are results of naturally trained models, and the last three rows are results of adversarially trained models.

	Target models			
Attack	Baseline	SphereConv	BallConv	TanhConv
None	85.35	88.58	91.13	**91.45**
FGSM	18.82	43.64	50.47	**52.60**
BIM	8.67	8.89	7.74	**10.18**
None	83.70	87.41	87.47	**87.54**
FGSM	78.96	**85.98**	82.20	81.46
BIM	7.96	**35.07**	17.38	19.86

Table 4.8: Black-box attacks on CIFAR-10. Performance is measured in accuracy (%). The first three rows are results of naturally trained models, and the last three rows are results of adversarially trained models.

	Target models			
Attack	Baseline	SphereConv	BallConv	TanhConv
None	85.35	88.58	91.13	**91.45**
FGSM	50.90	**56.71**	49.50	50.61
BIM	36.22	**43.10**	27.48	29.06
None	83.70	87.41	87.47	**87.54**
FGSM	77.57	76.29	78.67	**80.38**
BIM	78.55	77.79	80.59	**82.47**

Black-box Adversarial Attacks

We run black-box attacks on naturally-trained and FGSM-trained models on CIFAR-10 (see Table 4.8). With natural training, it is surprising that BallConv and TanhConv do not show an advantage over the baseline, while SphereConv performs the best. The strongly nonlinear landscape of BallConv and TanhConv may be too difficult to be optimized without adversarial training. SphereConv, with a tighter geometric constraint, is able to withstand adversarial attacks without adversarial training. With adversarial training, SphereConv is less resistant against adversarial attacks than the baseline. BallConv and TanhConv, however, show significant advantage over the baseline. Our observation that adversarial training compromises the robustness of SphereConv matches the conclusion made by [97]. Since SphereConv enforces a tight constraint of output vectors, the landscape around some data points will be dramatically changed during adversarial train-

ing. BallConv and TanhConv are less constrained and thus can fit adversarial examples without detrimental changes in the landscapes.

Part III:

Regularizations on Hypersphere

In this part, we revisit the regularizations in neural networks and propose novel weight regularizations that are not only more suitable for hyperspherical learning but also effective in standard neural networks. The widely used weight decay regularization is no longer useful in hyperspherical neural networks, because all the neuron norms will not affect the output activations. Therefore, we need to consider a weight regularization that aims to regularize the neuron direction instead of norm. Specifically, we introduce a concept of hyperspherical energy that characterizes angular diversity of neurons on the hypersphere. Then we propose to minimize this energy to regularize the neurons. Our experiments show that these regularizations can effectively improve the generalization.

Chapter 5 is based on the following publication:

- *W. Liu*, R. Lin, Z. Liu, L. Liu, Z. Yu, B. Dai, L. Song. Learning towards Minimum Hyperspherical Energy. *NeurIPS 2018*

Chapter 6 is based on the following publication:

- R. Lin, *W. Liu*, Z. Liu, C. Feng, Z. Yu, J. Rehg, L. Xiong, L. Song. Regularizing Neural Networks via Minimizing Hyperspherical Energy. *CVPR 2020*

CHAPTER 5
LEARNING TOWARDS MINIMUM HYPERSPHERICAL ENERGY

5.1 Introduction

The recent success of deep neural networks has led to its wide applications in a variety of tasks. With the over-parametrization nature and deep layered architecture, current deep networks [2, 60, 42] are able to achieve impressive performance on large-scale problems. Despite such success, having redundant and highly correlated neurons (*e.g.*, weights of kernels/filters in convolutional neural networks (CNNs)) caused by over-parametrization presents an issue [98, 99], which motivated a series of influential works in network compression [100, 101] and parameter-efficient network architectures [102, 103, 104]. These works either compress the network by pruning redundant neurons or directly modify the network architecture, aiming to achieve comparable performance while using fewer parameters. Yet, it remains an open problem to find a unified and principled theory that guides the network compression in the context of optimal generalization ability.

Another stream of works seeks to further release the network generalization power by alleviating redundancy through diversification [105, 106, 26, 27] as rigorously analyzed by [107]. Most of these works address the redundancy problem by enforcing relatively large diversity between pairwise projection bases via regularization. Our work broadly falls into this category by sharing similar high-level target, but the spirit and motivation behind our proposed models are distinct. In particular, there is a recent trend of studies that feature the significance of angular learning at both loss and convolution levels [9, 10, 12, 20], based on the observation that the angles in deep embeddings learned by CNNs tend to encode semantic difference. The key intuition is that angles preserve the most abundant and discriminative information for visual recognition. As a result, hyperspherical geodesic

distances between neurons naturally play a key role in this context, and thus, it is intuitively desired to impose discrimination by keeping their projections on the hypersphere as far away from each other as possible. While the concept of imposing large angular diversities was also considered in [107, 105, 106, 27], they do not consider diversity in terms of global equidistribution of embeddings on the hypersphere, which fails to achieve the state-of-the-art performances.

Given the above motivation, we draw inspiration from a well-known physics problem, called Thomson problem [108, 109]. The goal of Thomson problem is to determine the minimum electrostatic potential energy configuration of N mutually-repelling electrons on the surface of a unit sphere. We identify the intrinsic resemblance between the Thomson problem and our target, in the sense that diversifying neurons can be seen as searching for an optimal configuration of electron locations. Similarly, we characterize the diversity for a group of neurons by defining a generic hyperspherical potential energy using their pair-wise relationship. Higher energy implies higher redundancy, while lower energy indicates that these neurons are more diverse and more uniformly spaced. To reduce the redundancy of neurons and improve the neural networks, we propose a novel *minimum hyperspherical energy* (MHE) regularization framework, where the diversity of neurons is promoted by minimizing the hyperspherical energy in each layer. As verified by comprehensive experiments on multiple tasks, MHE is able to consistently improve the generalization power of neural networks.

MHE faces different situations when it is applied to hidden layers and output layers. For hidden layers, applying MHE straightforwardly may still encourage some degree of redundancy since it will produce co-linear bases pointing to opposite directions (see Figure 5.1 middle). In order to avoid such redundancy, we propose the half-space MHE which constructs a group of virtual neurons and minimize the hyperspherical energy of both existing and virtual neurons. For output layers, MHE aims to distribute the classifier neurons[1] as

[1]Classifier neurons are the projection bases of the last layer (*i.e.*, output layer) before input to softmax.

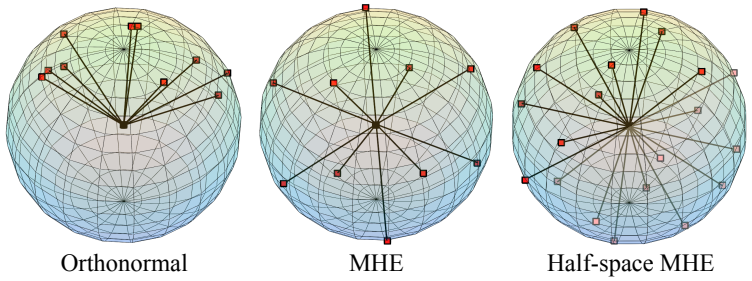


Figure 5.1: Orthonormal, MHE and half-space MHE regularization. The red dots denote the neurons optimized by the gradient of the corresponding regularization. The rightmost pink dots denote the virtual negative neurons. We randomly initialize the weights of 10 neurons on a 3D Sphere and optimize them with SGD.

uniformly as possible to improve the inter-class feature separability. Different from MHE in hidden layers, classifier neurons should be distributed in the full space for the best classification performance [9, 10]. An intuitive comparison among the widely used orthonormal regularization, the proposed MHE and half-space MHE is provided in Figure 5.1. One can observe that both MHE and half-space MHE are able to uniformly distribute the neurons over the hypersphere and half-space hypersphere, respectively. In contrast, conventional orthonormal regularization tends to group neurons closer, especially when the number of neurons is greater than the dimension.

MHE is originally defined on Euclidean distance, as indicated in Thomson problem. However, we further consider minimizing hyperspherical energy defined with respect to angular distance, which we will refer to as angular-MHE (A-MHE) in the following paper. In addition, we give some theoretical insights of MHE regularization, by discussing the asymptotic behavior and generalization error. Last, we apply MHE regularization to multiple vision tasks, including generic object recognition, class-imbalance learning, and face recognition. In the experiments, we show that MHE is architecture-agnostic and can considerably improve the generalization ability.

5.2 Related Works

Diversity regularization is shown useful in sparse coding [110, 111], ensemble learning [112, 113], self-paced learning [114], metric learning [115], etc. Early studies in sparse coding [110, 111] show that the generalization ability of codebook can be improved via diversity regularization, where the diversity is often modeled using the (empirical) covariance matrix. More recently, a series of studies have featured diversity regularization in neural networks [107, 105, 106, 26, 27, 28], where regularization is mostly achieved via promoting large angle/orthogonality, or reducing covariance between bases. Our work differs from these studies by formulating the diversity of neurons on the entire hypersphere, therefore promoting diversity from a more global, top-down perspective.

Methods other than diversity-promoting regularization have been widely proposed to improve CNNs [116, 44, 30, 12] and generative adversarial nets (GANs) [117, 118]. MHE can be regarded as a complement that can be applied on top of these methods.

5.3 Learning Neurons towards Minimum Hyperspherical Energy

5.3.1 Formulation of Minimum Hyperspherical Energy

Minimum hyperspherical energy defines an equilibrium state of the configuration of neuron's directions. We argue that the power of neural representation of each layer can be characterized by the hyperspherical energy of its neurons, and therefore a minimal energy configuration of neurons can induce better generalization. Before delving into details, we first define the hyperspherical energy functional for N neurons (*i.e.*, kernels) of $(d+1)$-dimension $\boldsymbol{W}_N = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_N \in \mathbb{R}^{d+1}\}$ as

$$\boldsymbol{E}_{s,d}(\hat{\boldsymbol{w}}_i|_{i=1}^N) = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} f_s\big(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|\big) = \begin{cases} \sum_{i\neq j} \|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-s}, & s > 0 \\ \sum_{i\neq j} \log\big(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-1}\big), & s = 0 \end{cases}, \tag{5.1}$$

where $\|\cdot\|$ denotes Euclidean distance, $f_s(\cdot)$ is a decreasing real-valued function, and $\hat{\boldsymbol{w}}_i = \frac{\boldsymbol{w}_i}{\|\boldsymbol{w}_i\|}$ is the i-th neuron weight projected onto the unit hypersphere $\mathbb{S}^d = \{\boldsymbol{w} \in \mathbb{R}^{d+1} | \, \|\boldsymbol{w}\| = 1\}$. We also denote $\hat{\boldsymbol{W}}_N = \{\hat{\boldsymbol{w}}_1, \cdots, \hat{\boldsymbol{w}}_N \in \mathbb{S}^d\}$, and $\boldsymbol{E}_s = \boldsymbol{E}_{s,d}(\hat{\boldsymbol{w}}_i|_{i=1}^N)$ for short. There are plenty of choices for $f_s(\cdot)$, but in this paper we use $f_s(z) = z^{-s}, s > 0$, known as Riesz s-kernels. Particularly, as $s \to 0$, $z^{-s} \to s\log(z^{-1}) + 1$, which is an affine transformation of $\log(z^{-1})$. It follows that optimizing the logarithmic hyperspherical energy $\boldsymbol{E}_0 = \sum_{i \neq j} \log(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-1})$ is essentially the limiting case of optimizing the hyperspherical energy $\boldsymbol{E}_s$. We therefore define $f_0(z) = \log(z^{-1})$ for convenience.

The goal of the MHE criterion is to minimize the energy in Equation 5.1 by varying the orientations of the neuron weights $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_N$. To be precise, we solve an optimization problem: $\min_{\boldsymbol{W}_N} \boldsymbol{E}_s$ with $s \geq 0$. In particular, when $s = 0$, we solve the logarithmic energy minimization problem:

$$\arg\min_{\boldsymbol{W}_N} \boldsymbol{E}_0 = \arg\min_{\boldsymbol{W}_N} \exp(\boldsymbol{E}_0) = \arg\max_{\boldsymbol{W}_N} \prod_{i \neq j} \|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|, \tag{5.2}$$

in which we essentially maximize the product of Euclidean distances. $\boldsymbol{E}_0$, $\boldsymbol{E}_1$ and $\boldsymbol{E}_2$ have interesting yet profound connections. Note that Thomson problem corresponds to minimizing $\boldsymbol{E}_1$, which is a NP-hard problem. Therefore in practice we can only compute its approximate solution by heuristics. In neural networks, such a differentiable objective can be directly optimized via gradient descent.

5.3.2 Logarithmic Hyperspherical Energy $\boldsymbol{E}_0$ as a Relaxation

Optimizing the original energy in Equation 5.1 is equivalent to optimizing its logarithmic form $\log \boldsymbol{E}_s$. To efficiently solve this difficult optimization problem, we can instead optimize the lower bound of $\log \boldsymbol{E}_s$ as a surrogate energy, by applying Jensen's inequality:

$$\arg\min_{\boldsymbol{W}_N} \left\{ \boldsymbol{E}_{\log} := \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \log\Big(f_s\big(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\| \big) \Big) \right\} \tag{5.3}$$

With $f_s(z) = z^{-s}, s > 0$, we observe that $\boldsymbol{E}_{\log}$ becomes $s\boldsymbol{E}_0 = s\sum_{i\neq j}\log(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-1})$, which is identical to the logarithmic hyperspherical energy $\boldsymbol{E}_0$ up to a multiplicative factor s. Therefore, minimizing $\boldsymbol{E}_0$ can also be viewed as a relaxation of minimizing $\boldsymbol{E}_s$ for $s > 0$.

5.3.3 MHE as Regularization for Neural Networks

Now that we have introduced the formulation of MHE, we propose MHE regularization for neural networks. In supervised neural network learning, the entire objective function is shown as follows:

$$\mathcal{L} = \underbrace{\frac{1}{m}\sum_{j=1}^{m}\ell(\langle \boldsymbol{w}_i^{\text{out}}, \boldsymbol{x}_j\rangle_{i=1}^{c}, \boldsymbol{y}_j)}_{\text{training data fitting}} + \underbrace{\lambda_{\text{h}}\cdot\sum_{j=1}^{L-1}\frac{1}{N_j(N_j-1)}\{\boldsymbol{E}_s\}_j}_{T_{\text{h}}\text{: hyperspherical energy for hidden layers}} + \underbrace{\lambda_{\text{o}}\cdot\frac{1}{N_L(N_L-1)}\boldsymbol{E}_s(\hat{\boldsymbol{w}}_i^{\text{out}}|_{i=1}^{c})}_{T_{\text{o}}\text{: hyperspherical energy for output layer}} \tag{5.4}$$

where $\boldsymbol{x}_i$ is the feature of the i-th training sample entering the output layer, $\boldsymbol{w}_i^{\text{out}}$ is the classifier neuron for the i-th class in the output fully-connected layer and $\hat{\boldsymbol{w}}_i^{\text{out}}$ denotes its normalized version. $\{\boldsymbol{E}_s\}_i$ denotes the hyperspherical energy for the neurons in the i-th layer. c is the number of classes, m is the batch size, L is the number of layers of the neural network, and N_i is the number of neurons in the i-th layer. $\boldsymbol{E}_s(\hat{\boldsymbol{w}}_i^{\text{out}}|_{i=1}^{c})$ denotes the hyperspherical energy of neurons $\{\hat{\boldsymbol{w}}_1^{\text{out}}, \cdots, \hat{\boldsymbol{w}}_c^{\text{out}}\}$. The ℓ_2 weight decay is omitted here for simplicity, but we will use it in practice. An alternative interpretation of MHE regularization from a decoupled view is given in subsection 5.3.7 and section D.3. MHE has different effects and interpretations in regularizing hidden layers and output layers.

MHE for hidden layers. To make neurons in the hidden layers more discriminative and less redundant, we propose to use MHE as a form of regularization. MHE encourages the normalized neurons to be uniformly distributed on a unit hypersphere, which is

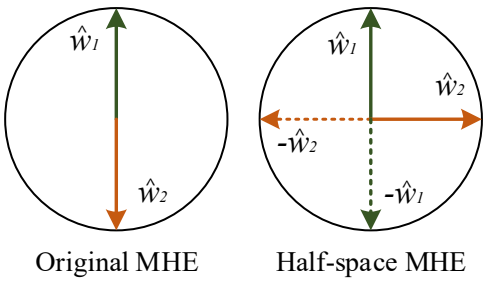


Figure 5.2: Half-space MHE.

partially inspired by the observation in [12] that angular difference in neurons preserves semantic (label-related) information. To some extent, MHE maximizes the average angular difference between neurons (specifically, the hyperspherical energy of neurons in every hidden layer). For instance, in CNNs we minimize the hyperspherical energy of kernels in convolutional and fully-connected layers except the output layer.

MHE for output layers. For the output layer, we propose to enhance the inter-class feature separability with MHE to learn discriminative and well-separated features. For classification tasks, MHE regularization is complementary to the softmax cross-entropy loss in CNNs. The softmax loss focuses more on the intra-class compactness, while MHE encourages the inter-class separability. Therefore, MHE on output layers can induce features with better generalization power.

5.3.4 MHE in Half Space

Directly applying the MHE formulation may still encounter some redundancy. An example in Figure 5.2, with two neurons in a 2-dimensional space, illustrates this potential issue. Directly imposing the original MHE regularization leads to a solution that two neurons are collinear but with opposite directions. To avoid such redundancy, we propose the half-space MHE regularization which constructs some virtual neurons and minimizes the hyperspherical energy of both original and virtual neurons together. Specifically, half-space MHE constructs a collinear virtual neuron with opposite direction for every existing neuron. Therefore, we end up with minimizing the hyperspherical energy with $2N_i$ neurons in

the i-th layer (*i.e.*, minimizing $\boldsymbol{E}_s(\{\hat{\boldsymbol{w}}_k, -\hat{\boldsymbol{w}}_k\}|_{k=1}^{2N_i})$). This half-space variant will encourage the neurons to be less correlated and less redundant, as illustrated in Figure 5.2. Note that, half-space MHE can only be used in hidden layers, because the collinear neurons do not constitute redundancy in output layers, as shown in [9]. Nevertheless, collinearity is usually not likely to happen in high-dimensional spaces, especially when the neurons are optimized to fit training data. This may be the reason that the original MHE regularization still consistently improves the baselines.

5.3.5 MHE beyond Euclidean Distance

The hyperspherical energy is originally defined based on the Euclidean distance on a hypersphere, which can be viewed as an angular measure. In addition to Euclidean distance, we further consider the geodesic distance on a unit hypersphere as a distance measure for neurons, which is exactly the same as the angle between neurons. Specifically, we consider to use $\arccos(\hat{\boldsymbol{w}}_i^\top \hat{\boldsymbol{w}}_j)$ to replace $\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|$ in hyperspherical energies. Following this idea, we propose angular MHE (A-MHE) as a simple extension, where the hyperspherical energy is rewritten as:

$$\boldsymbol{E}_{s,d}^a(\hat{\boldsymbol{w}}_i|_{i=1}^N) = \sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s\big(\arccos(\hat{\boldsymbol{w}}_i^\top \hat{\boldsymbol{w}}_j)\big) = \begin{cases} \sum_{i\neq j} \arccos(\hat{\boldsymbol{w}}_i^\top \hat{\boldsymbol{w}}_j)^{-s}, & s > 0 \\ \sum_{i\neq j} \log\big(\arccos(\hat{\boldsymbol{w}}_i^\top \hat{\boldsymbol{w}}_j)^{-1}\big), & s = 0 \end{cases} \tag{5.5}$$

which can be viewed as redefining MHE based on geodesic distance on hyperspheres (*i.e.*, angle), and can be used as an alternative to the original hyperspherical energy $\boldsymbol{E}_s$ in Equation 5.4. Note that, A-MHE can also be learned in full-space or half-space, leading to similar variants as original MHE. The key difference between MHE and A-MHE lies in the optimization dynamics, because their gradients w.r.t the neuron weights are quite different. A-MHE is also more computationally expensive than MHE.

5.3.6 Mini-batch Approximation for MHE

With a large number of neurons in one layer, calculating MHE can be computationally expensive as it requires computing the pair-wise distances between neurons. To address this issue, we propose the mini-batch version of MHE to approximate the MHE (either original or half-space) objective.

Mini-batch approximation for MHE on hidden layers. For hidden layers, mini-batch approximation iteratively takes a random batch of neurons as input and minimizes their hyperspherical energy as an approximation to the MHE. Note that the gradient of the mini-batch objective is an effective estimation of the original gradient of MHE.

Data-dependent mini-batch approximation for output layers. For the output layer, the data-dependent mini-batch approximation will iteratively takes the classifier neurons corresponding to the classes that appear in the mini-batches to compute their hyperspherical energy. It minimizes the objective $\frac{1}{m(N-1)}\sum_{i=1}^{m}\sum_{j=1,j\neq y_i}^{N} f_s(\|\hat{\boldsymbol{w}}_{y_i} - \hat{\boldsymbol{w}}_j\|)$ in each iteration, where y_i denotes the class label of the i-th sample in each mini-batch, m is the mini-batch size, and N is the number of neurons (in one particular layer).

5.3.7 Discussions

Connections to scientific problems. The hyperspherical energy minimization has close relationships with scientific problems. When $s = 1$, Equation 5.1 reduces to Thomson problem [108, 109] (in physics) where one needs to determine the minimum electrostatic potential energy configuration of N mutually-repelling electrons on a unit sphere. When $s = \infty$, Equation 5.1 becomes Tammes problem [119] (in geometry) where the goal is to pack a given number of circles on the surface of a sphere such that the minimum distance between circles is maximized. When $s = 0$, Equation 5.1 becomes Whyte's problem where the goal is to maximize product of Euclidean distances as shown in Equation 5.2. Our work aims to make use of important insights from these scientific problems to improve neural networks.

Understanding MHE from decoupled view. Inspired by decoupled networks [20], we can view the original convolution as the multiplication of the angular function $g(\theta) = \cos(\theta)$ and the magnitude function $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \|\boldsymbol{w}\| \cdot \|\boldsymbol{x}\|$: $f(\boldsymbol{w}, \boldsymbol{x}) = h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) \cdot g(\theta)$ where θ is the angle between the kernel $\boldsymbol{w}$ and the input $\boldsymbol{x}$. From the equation above, we can see that the norm of the kernel and the direction (*i.e.*, angle) of the kernel affect the inner product similarity differently. Typically, weight decay is to regularize the kernel by minimizing its ℓ_2 norm, while there is no regularization on the direction of the kernel. Therefore, MHE completes this missing piece by promoting angular diversity. By combining MHE to a standard neural networks, the entire regularization term becomes

$$\mathcal{L}_{\text{reg}} = \underbrace{\lambda_{\text{w}} \cdot \frac{1}{\sum_{j=1}^{L} N_j} \sum_{j=1}^{L} \sum_{i=1}^{N_j} \|\boldsymbol{w}_i\|}_{\text{Weight decay: regularizing the magnitude of kernels}} + \underbrace{\lambda_{\text{h}} \cdot \sum_{j=1}^{L-1} \frac{1}{N_j(N_j-1)} \{\boldsymbol{E}_s\}_j + \lambda_{\text{o}} \cdot \frac{1}{N_L(N_L-1)} \boldsymbol{E}_s(\hat{\boldsymbol{w}}_i^{\text{out}}|_{i=1}^{c})}_{\text{MHE: regularizing the direction of kernels}}$$

where λ_{w}, λ_{h} and λ_{o} are weighting hyperparameters for these three regularization terms. From the decoupled view, MHE makes a lot of senses in regularizing the neural networks, since it serves as a complementary and orthogonal role to weight decay. More discussions are in section D.3.

Comparison to orthogonality/angle-promoting regularizations. Promoting orthogonality or large angles between bases has been a popular choice for encouraging diversity. Probably the most related and widely used one is the orthonormal regularization [12] which aims to minimize $\|\boldsymbol{W}^\top \boldsymbol{W} - \boldsymbol{I}\|_F$, where W denotes the weights of a group of neurons with each column being one neuron and $\boldsymbol{I}$ is an identity matrix. One similar regularization is the orthogonality regularization [27] which minimizes the sum of the cosine values between all the kernel weights. These methods encourage kernels to be orthogonal to each other, while MHE does not. Instead, MHE encourages the hyperspherical diversity among these

kernels, and these kernels are not necessarily orthogonal to each other. [106] proposes the angular constraint which aims to constrain the angles between different kernels of the neural network, but quite different from MHE, they use a hard constraint to impose this angular regularization. Moreover, these methods model diversity regularization at a more local level, while MHE regularization seeks to model the problem in a more top-down manner.

Normalized neurons in MHE. From Equation 5.1, one can see that the normalized neurons are used to compute MHE, because we aim to encourage the diversity on a hypersphere. However, a natural question may arise: what if we use the original (*i.e.*, unnormalized) neurons to compute MHE? First, combining the norm of kernels (*i.e.*, neurons) into MHE may lead to a trivial gradient descent direction: simply increasing the norm of all kernels. Suppose all kernel directions stay unchanged, increasing the norm of all kernels by a factor can effectively decrease the objective value of MHE. Second, coupling the norm of kernels into MHE may contradict with weight decay which aims to decrease the norm of kernels. Moreover, normalized neurons imply that the importance of all neurons is the same, which matches the intuition in [10, 12, 20]. If we desire different importance for different neurons, we can also manually assign a fixed weight for each neuron. This may be useful when we have already known certain neurons are more important and we want them to be relatively fixed. The neuron with large weight tends to be updated less. We will discuss it more in section D.4.

5.4 Theoretical Insights

This section leverages a number of rigorous theoretical results from [120, 121, 122, 123, 124, 121, 125, 126] and provides theoretical yet intuitive understandings about MHE.

5.4.1 Asymptotic Behavior

This subsection shows how the hyperspherical energy behaves asymptotically. Specifically, as $N \to \infty$, we can show that the solution $\hat{\boldsymbol{W}}_N$ tends to be uniformly distributed on hypersphere $\mathbb{S}^d$ when the hyperspherical energy defined in Equation 5.1 achieves its minimum.

Definition 2 (minimal hyperspherical s-energy)**.** *We define the minimal s-energy for N points on the unit hypersphere $\mathbb{S}^d = \{w \in \mathbb{R}^{d+1} | \, \|\boldsymbol{w}\| = 1\}$ as*

$$\boldsymbol{\varepsilon}_{s,d}(N) := \inf_{\hat{\boldsymbol{W}}_N \subset \mathbb{S}^d} \boldsymbol{E}_{s,d}(\hat{\boldsymbol{w}}_i|_{i=1}^N) \tag{5.6}$$

where the infimum is taken over all possible $\hat{\boldsymbol{W}}_N$ on $\mathbb{S}^d$. Any configuration of $\hat{\boldsymbol{W}}_N$ to attain the infimum is called an s-extremal configuration. Usually $\boldsymbol{\varepsilon}_{s,d}(N) = \infty$ if N is greater than d and $\boldsymbol{\varepsilon}_{s,d}(N) = 0$ if $N = 0, 1$.

We discuss the asymptotic behavior ($N \to \infty$) in three cases: $0 < s < d$, $s = d$, and $s > d$. We first write the energy integral as $I_s(\mu) = \iint_{\mathbb{S}^d \times \mathbb{S}^d} \|\boldsymbol{u} - \boldsymbol{v}\|^{-s} d\mu(\boldsymbol{u}) d\mu(\boldsymbol{v})$, which is taken over all probability measure μ supported on $\mathbb{S}^d$. With $0 < s < d$, $I_s(\mu)$ is minimal when μ is the spherical measure $\sigma^d = \mathcal{H}^d(\cdot)|_{\mathbb{S}^d} / \mathcal{H}^d(\mathbb{S}^d)$ on $\mathbb{S}^d$, where $\mathcal{H}^d(\cdot)$ denotes the d-dimensional Hausdorff measure. When $s \geq d$, $I_s(\mu)$ becomes infinity, which therefore requires different analysis. In general, we can say all s-extremal configurations asymptotically converge to uniform distribution on a hypersphere, as stated in Theorem 1. This asymptotic behavior has been heavily studied in [120, 121, 122].

Theorem 1 (asymptotic uniform distribution on hypersphere)**.** *Any sequence of optimal s-energy configurations $(\hat{\boldsymbol{W}}_N^\star)|_2^\infty \subset \mathbb{S}^d$ is asymptotically uniformly distributed on $\mathbb{S}^d$ in the sense of the weak-star topology of measures, namely*

$$\frac{1}{N} \sum_{\boldsymbol{v} \in \hat{\boldsymbol{W}}_N^\star} \delta_{\boldsymbol{v}} \to \sigma^d, \qquad \text{as } N \to \infty \tag{5.7}$$

where $\delta_{\boldsymbol{v}}$ denotes the unit point mass at $\boldsymbol{v}$, and σ^d is the spherical measure on $\mathbb{S}^d$.

Theorem 2 (asymptotics of the minimal hyperspherical s-energy)**.** *We have that* $\lim_{N\to\infty} \frac{\mathcal{E}_{s,d}(N)}{p(N)}$ *exists for the minimal s-energy. For* $0 < s < d$*, $p(N) = N^2$. For $s = d$, $p(N) = N^2 \log N$. For $s > d$, $p(N) = N^{1+s/d}$. Particularly if* $0 < s < d$*, we have* $\lim_{N\to\infty} \frac{\mathcal{E}_{s,d}(N)}{N^2} = I_s(\sigma^d)$.

Theorem Theorem 2 tells us the growth power of the minimal hyperspherical s-energy when N goes to infinity. Therefore, different potential power s leads to different optimization dynamics. In the light of the behavior of the energy integral, MHE regularization will focus more on local influence from neighborhood neurons instead of global influences from all the neurons as the power s increases.

5.4.2 Generalization and Optimization

As proved in [126], in one-hidden-layer neural network, the diversity of neurons can effectively eliminate the spurious local minima despite the non-convexity in learning dynamics of neural networks. Following such an argument, our MHE regularization, which encourages the diversity of neurons, naturally matches the theoretical intuition in [126], and effectively promotes the generalization of neural networks. While hyperspherical energy is minimized such that neurons become diverse on hyperspheres, the hyperspherical diversity is closely related to the generalization error.

More specifically, in a one-hidden-layer neural network $f(x) = \sum_{k=1}^n v_k \sigma(\boldsymbol{W}_k^\top \boldsymbol{x})$ with least squares loss $L(f) = \frac{1}{2m}\sum_{i=1}^m (y_i - f(\boldsymbol{x}_i))^2$, we can compute its gradient w.r.t $\boldsymbol{W}_k$ as $\frac{\partial L}{\partial \boldsymbol{W}_k} = \frac{1}{m}\sum_{i=1}^m (f(x_i) - y_i) v_k \sigma'(\boldsymbol{W}_k^\top \boldsymbol{x}_i)\boldsymbol{x}_i$. ($\sigma(\cdot)$ is the nonlinear activation function and $\sigma'(\cdot)$ is its subgradient. $\boldsymbol{x} \in$ is the training sample. $\boldsymbol{W}_k$ denotes the weights of hidden layer and v_k is the weights of output layer.) Subsequently, we can rewrite this gradient as a matrix form: $\frac{\partial L}{\partial \boldsymbol{W}} = \boldsymbol{D} \cdot \boldsymbol{r}$ where $\boldsymbol{D} \in \mathbb{R}^{dn\times m}, \boldsymbol{D}_{\{di-d+1:di,j\}} = v_i \sigma'(\boldsymbol{W}_i^\top \boldsymbol{x}_j)\boldsymbol{x}_j \in \mathbb{R}^d$ and $\boldsymbol{r} \in \mathbb{R}^m, \boldsymbol{r}_i = \frac{1}{m} f(\boldsymbol{x}_i) - y_i$. Further, we can obtain the inequality $\|\boldsymbol{r}\| \leq \frac{1}{\lambda_{\min}(\boldsymbol{D})} \|\frac{\partial L}{\partial \boldsymbol{W}}\|$. $\|\boldsymbol{r}\|$ is actually the training error. To make the training error small, we need to lower bound $\lambda_{\min}(\boldsymbol{D})$ away from zero. From [126, 127], one can know that the lower bound of $\lambda_{\min}(\boldsymbol{D})$

Table 5.1: Testing error (%) of different MHE on CIFAR-10/100.

Method	CIFAR-10			CIFAR-100		
	$s=2$	$s=1$	$s=0$	$s=2$	$s=1$	$s=0$
MHE	6.22	6.74	6.44	27.15	27.09	**26.16**
Half-space MHE	6.28	6.54	**6.30**	**25.61**	**26.30**	26.18
A-MHE	**6.21**	6.77	6.45	26.17	27.31	27.90
Half-space A-MHE	6.52	**6.49**	6.44	26.03	26.52	26.47
Baseline	7.75			28.13		

is directly related to the hyperspherical diversity of neurons. After bounding the training error, it is easy to bound the generalization error using Rademachar complexity.

5.5 Applications and Experiments

5.5.1 Improving Network Generalization

First, we perform ablation study and some exploratory experiments on MHE. Then we apply MHE to large-scale object recognition and class-imbalance learning. For all the experiments on CIFAR-10 and CIFAR-100 in the paper, we use moderate data augmentation, following [2, 20]. For ImageNet-2012, we follow the same data augmentation in [12]. We train all the networks using SGD with momentum 0.9, and the network initialization follows [38]. All the networks use BN [44] and ReLU if not otherwise specified. Experimental details are given in each subsection and section D.1.

Ablation Study and Exploratory Experiments

Variants of MHE. We evaluate all different variants of MHE on CIFAR-10 and CIFAR-100, including original MHE (with the power $s = 0, 1, 2$) and half-space MHE (with the power $s = 0, 1, 2$) with both Euclidean and angular distance. In this experiment, all methods use CNN-9 (see section D.1). The results in Table 5.1 show that all the variants of MHE perform consistently better than the baseline. Specifically, the half-space MHE has more significant performance gain compared to the other MHE variants, and MHE with Euclidean and angular distance perform similarly. In general, MHE with $s = 2$ performs

Table 5.2: Testing error (%) of different width on CIFAR-100.

Method	16/32/64	32/64/128	64/128/256	128/256/512	256/512/1024
Baseline	47.72	38.64	28.13	24.95	25.45
MHE	36.84	30.05	26.75	24.05	23.14
Half-space MHE	**35.16**	**29.33**	**25.96**	**23.38**	**21.83**

Table 5.3: Testing error (%) of different depth on CIFAR-100. N/C: not converged.

Method	CNN-6	CNN-9	CNN-15
Baseline	32.08	28.13	N/C
MHE	28.16	26.75	26.9
Half-space MHE	**27.56**	**25.96**	**25.84**

best among $s = 0, 1, 2$. In the following experiments, we use $s = 2$ and Euclidean distance for both MHE and half-space MHE by default if not otherwise specified.

Network width. We evaluate MHE with different network width. We use CNN-9 as our base network, and change its filter number in Conv1.x, Conv2.x and Conv3.x (see section D.1) to 16/32/64, 32/64/128, 64/128/256, 128/256/512 and 256/512/1024. Results in Table 5.2 show that both MHE and half-space MHE consistently outperform the baseline, showing stronger generalization. Interestingly, both MHE and half-space MHE have more significant gain while the filter number is smaller in each layer, indicating that MHE can help the network to make better use of the neurons. In general, half-space MHE performs consistently better than MHE, showing the necessity of reducing collinearity redundancy among neurons. Both MHE and half-space MHE outperform the baseline with a huge margin while the network is either very wide or very narrow, showing the superiority in improving generalization.

Network depth. We perform experiments with different network depth to better evaluate the performance of MHE. We fix the filter number in Conv1.x, Conv2.x and Conv3.x to 64, 128 and 256, respectively. We compare 6-layer CNN, 9-layer CNN and 15-layer CNN. The results are given in Table 5.3. Both MHE and half-space MHE perform significantly better than the baseline. More interestingly, baseline CNN-15 can not converge, while CNN-15 is able to converge reasonably well if we use MHE to regularize the network.

Table 5.4: Ablation study on CIFAR-100.

Method	H O × √	H O √ ×	H O √ √
MHE	26.85	26.55	**26.16**
Half-space MHE	N/A	26.28	**25.61**
A-MHE	27.8	26.56	**26.17**
Half-space A-MHE	N/A	26.64	**26.03**
Baseline	28.13		

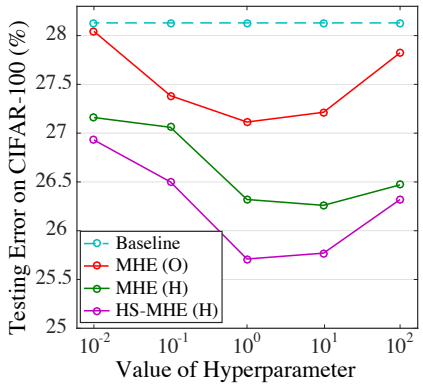


Figure 5.3: Effect of hyperparameter.

Moreover, we also see that half-space MHE can consistently show better generalization than MHE with different network depth.

Ablation study. Since the current MHE regularizes the neurons in the hidden layers and the output layer simultaneously, we perform ablation study for MHE to further investigate where the gain comes from. This experiment uses the CNN-9. The results are given in Table 5.4. "H" means that we apply MHE to all the hidden layers, while "O" means that we apply MHE to the output layer. Because the half-space MHE can not be applied to the output layer, so there is "N/A" in the table. In general, we find that applying MHE to both the hidden layers and the output layer yields the best performance, and using MHE in the hidden layers usually produces better accuracy than using MHE in the output layer.

Hyperparameter experiment. We evaluate how the selection of hyperparameter affects the performance. We experiment with different hyperparameters from 10^{-2} to 10^{2} on CIFAR-100 with the CNN-9. HS-MHE denotes the half-space MHE. We evaluate MHE variants by separately applying MHE to the output layer ("O"), MHE to the hidden layers

Table 5.5: Testing error (%) of ResNet-32 on CIFAR-10/100.

Method	CIFAR-10	CIFAR-100
ResNet-110-original [2]	6.61	25.16
ResNet-1001 [85]	4.92	**22.71**
ResNet-1001 (64 batch) [85]	**4.64**	-
baseline	5.19	22.87
MHE	4.72	22.19
Half-space MHE	**4.66**	**22.04**

("H"), and the half-space MHE to the hidden layers ("H"). The results in Figure 5.3 show that our MHE is not very hyperparameter-sensitive and can consistently beat the baseline by a considerable margin. One can observe that MHE's hyperparameter works well from 10^{-2} to 10^{2} and therefore is easy to set. In contrast, the hyperparameter of weight decay could be more sensitive than MHE. Half-space MHE can consistently outperform the original MHE under all different hyperparameter settings. Interestingly, applying MHE only to hidden layers can achieve better accuracy than applying MHE only to output layers.

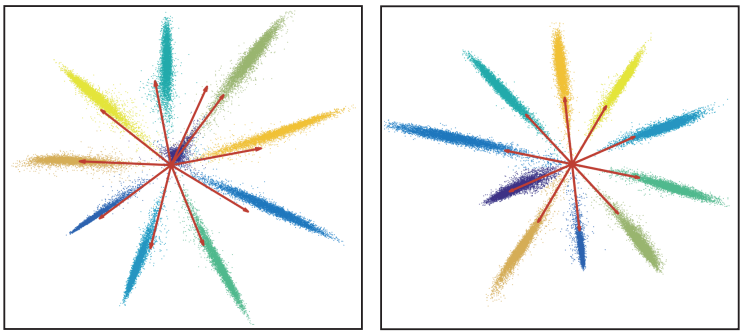
MHE for ResNets. Besides the standard CNN, we also evaluate MHE on ResNet-32 to show that our MHE is architecture-agnostic and can improve accuracy on multiple types of architectures. Besides ResNets, MHE can also be applied to GoogleNet [60], SphereNets [12] (the experimental results are given in section D.5), DenseNet [89], etc. Detailed architecture settings are given in section D.1. The results on CIFAR-10 and CIFAR-100 are given in Table 5.5. One can observe that applying MHE to ResNet also achieves considerable improvements, showing that MHE is generally useful for different architectures. Most importantly, adding MHE regularization will not affect the original architecture settings, and it can readily improve the network generalization at a neglectable computational cost.

Large-scale Object Recognition

We evaluate MHE on large-scale ImageNet-2012 datasets. Specifically, we perform experiment using ResNets, and then report the top-1 validation error (center crop) in Table 5.6. From the results, we still observe that both MHE and half-space MHE yield con-

Table 5.6: Top-1 error (%) on ImageNet.

Method	ResNet-18	ResNet-34
baseline	32.95	30.04
Orthogonal [27]	32.65	29.74
Orthonormal	32.61	29.75
MHE	32.50	29.60
Half-space MHE	**32.45**	**29.50**



(a) CNN without MHE (b) CNN with MHE

Figure 5.4: Class-imbalance learning on MNIST.

sistently better recognition accuracy than the baseline and the orthonormal regularization (after tuning its hyperparameter). To better evaluate the consistency of MHE's performance gain, we use two ResNets with different depth: ResNet-18 and ResNet-34. On these two different networks, both MHE and half-space MHE outperform the baseline by a significant margin, showing consistently better generalization power. Moreover, half-space MHE performs slightly better than full-space MHE as expected.

Class-imbalance Learning

Because MHE aims to maximize the hyperspherical margin between different classifier neurons in the output layer, we can naturally apply MHE to class-imbalance learning where the number of training samples in different classes is imbalanced. We demonstrate the power of MHE in class-imbalance learning through a toy experiment. We first randomly throw away 98% training data for digit 0 in MNIST (only 100 samples are preserved for digit 0), and then train a 6-layer CNN on this imbalance MNIST. To visualize the learned features, we set the output feature dimension as 2. The features and classifier neurons on the

Table 5.7: Testing error (%) on imbalanced CIFAR-10.

Method	Single	Err. (S)	Multiple
Baseline	9.80	30.40	12.00
Orthonormal	8.34	26.80	10.80
MHE	7.98	**25.80**	10.25
Half-space MHE	7.90	26.40	**9.59**
A-MHE	7.96	26.00	9.88
Half-space A-MHE	**7.59**	25.90	9.89

full training set are visualized in Figure 5.4 where each color denotes a digit and red arrows are the normalized classifier neurons. Although we train the network on the imbalanced training set, we visualize the features of the full training set for better demonstration. The visualization for the full testing set is also given in section D.8. From Figure 5.4, one can see that the CNN without MHE tends to ignore the imbalanced class (digit 0) and the learned classifier neuron is highly biased to another digit. In contrast, the CNN with MHE can learn reasonably separable distribution even if digit 0 only has 2% samples compared to the other classes. Using MHE in this toy setting can readily improve the accuracy on the full testing set from 88.5% to 98%. Most importantly, the classifier neuron for digit 0 is also properly learned, similar to the one learned on the balanced dataset. Note that, half-space MHE can not be applied to the classifier neurons, because the classifier neurons usually need to occupy the full feature space.

We experiment MHE in two data imbalance settings on CIFAR-10: 1) single class imbalance (S) - All classes have the same number of images but one single class has significantly less number, and 2) multiple class imbalance (M) - The number of images decreases as the class index decreases from 9 to 0. We use CNN-9 for all the compared regularizations. Detailed setups are provided in section D.1. In Table 5.7, we report the error rate on the whole testing set. In addition, we report the error rate (denoted by Err. (S)) on the imbalance class (single imbalance setting) in the full testing set. From the results, one can observe that CNN-9 with MHE is able to effectively perform recognition when classes are imbalanced. Even only given a small portion of training data in a few classes, CNN-9 with MHE can achieve very competitive accuracy on the full testing set, showing MHE's supe-

rior generalization power. Moreover, we also provide experimental results on imbalanced CIFAR-100 in section D.8.

5.5.2 SphereFace+: Improving Inter-class Feature Separability via MHE for Face Recognition

We have shown that full-space MHE for output layers can encourage classifier neurons to distribute more evenly on hypersphere and therefore improve inter-class feature separability. Intuitively, the classifier neurons serve as the approximate center for features from each class, and can therefore guide the feature learning. We also observe that open-set face recognition (*e.g.*, face verification) requires the feature centers to be as separable as possible [10]. This connection inspires us to apply MHE to face recognition. Specifically, we propose *SphereFace+* by applying MHE to SphereFace [10]. The objective of SphereFace, angular softmax loss (ℓ_{SF}) that encourages intra-class feature compactness, is naturally complementary to that of MHE. The objective function of SphereFace+ is

$$\begin{aligned}\mathcal{L}_{\text{SF+}} = \quad & \underbrace{\frac{1}{m}\sum_{j=1}^{m}\ell_{\text{SF}}(\langle \boldsymbol{w}_i^{\text{out}}, \boldsymbol{x}_j\rangle_{i=1}^{c}, \boldsymbol{y}_j, m_{\text{SF}})}_{\text{Angular softmax loss: promoting } \textbf{intra-class compactness}} \\ & \underbrace{+\lambda_{\text{M}}\cdot\frac{1}{m(N-1)}\sum_{i=1}^{m}\sum_{j=1,j\neq y_i}^{N} f_s(\left\|\hat{\boldsymbol{w}}_{y_i}^{\text{out}} - \hat{\boldsymbol{w}}_j^{\text{out}}\right\|)}_{\text{MHE: promoting } \textbf{inter-class separability}}\end{aligned} \tag{5.8}$$

where c is the number of classes, m is the mini-batch size, N is the number of classifier neurons, $\boldsymbol{x}_i$ the deep feature of the i-th face (y_i is its groundtruth label), and $\boldsymbol{w}_i^{\text{out}}$ is the i-th classifier neuron. m_{SF} is a hyperparameter for SphereFace, controlling the degree of intra-class feature compactness (*i.e.*, the size of the angular margin). Because face datesets usually have thousands of identities, we will use the data-dependent mini-batch approximation MHE as shown in Equation 5.8 in the output layer to reduce computational cost. MHE completes a missing piece for SphereFace by promoting the inter-class separability. SphereFace+ consistently outperforms SphereFace, and achieves state-of-the-art perfor-

Table 5.8: Testing accuracy (%) on the SphereFace-20 network.

m_{SF}	LFW		MegaFace	
	SphereFace	SphereFace+	SphereFace	SphereFace+
1	96.35	**97.15**	39.12	**45.90**
2	98.87	**99.05**	60.48	**68.51**
3	98.97	**99.13**	63.71	**66.89**
4	99.26	**99.32**	70.68	**71.30**

Table 5.9: Testing accuracy (%) on the SphereFace-64 network.

m_{SF}	LFW		MegaFace	
	SphereFace	SphereFace+	SphereFace	SphereFace+
1	96.93	**97.47**	41.07	**45.55**
2	99.03	**99.22**	62.01	**67.07**
3	99.25	**99.35**	69.69	**70.89**
4	99.42	**99.47**	72.72	**73.03**

mance on both LFW [41] and MegaFace [65] datasets. More results on MegaFace are put in section D.9. More evaluations and results can be found in section D.6 and section D.10.

Performance under different m_{SF}. We evaluate SphereFace+ with two different architectures (SphereFace-20 and SphereFace-64) proposed in [10]. Specifically, SphereFace-20 and SphereFace-64 are 20-layer and 64-layer modified residual networks, respectively. We train our network with the publicly available CASIA-Webface dataset [57], and then test the learned model on LFW and MegaFace dataset. In MegaFace dataset, the reported accuracy indicates rank-1 identification accuracy with 1 million distractors. All the results in Table 5.8 and Table 5.9 are computed without model ensemble and PCA. One can observe that SphereFace+ consistently outperforms SphereFace by a considerable margin on both LFW and MegaFace datasets under all different settings of m_{SF}. Moreover, the performance gain generalizes across network architectures with different depth.

Comparison to state-of-the-art methods. We also compare our methods with some widely used loss functions. All these compared methods use SphereFace-64 network that are trained with CASIA dataset. All the results are given in Table 5.10 computed without model ensemble and PCA. Compared to the other state-of-the-art methods, SphereFace+ achieves the best accuracy on LFW dataset, while being comparable to the best accuracy

Table 5.10: Comparison to the state-of-the-art methods on LFW and MegaFace.

Method	LFW	MegaFace
Softmax Loss	97.88	54.86
Softmax+Contrastive [36]	98.78	65.22
Triplet Loss [34]	98.70	64.80
L-Softmax Loss [9]	99.10	67.13
Softmax+Center Loss [63]	99.05	65.49
CosineFace [13, 14]	**99.10**	**75.10**
SphereFace	99.42	72.72
SphereFace+ (ours)	**99.47**	**73.03**

on MegaFace dataset. Current state-of-the-art face recognition methods [14, 10, 13, 15, 18] usually only focus on compressing the intra-class features, which makes MHE a potentially useful tool in order to further improve these face recognition methods.

5.6 Concluding Remarks

We borrow some useful ideas and insights from physics and propose a novel regularization method for neural networks, called minimum hyperspherical energy (MHE), to encourage the angular diversity of neuron weights. MHE can be easily applied to every layer of a neural network as a plug-in regularization, without modifying the original network architecture. Different from existing methods, such diversity can be viewed as uniform distribution over a hypersphere. In this paper, MHE has been specifically used to improve network generalization for generic image classification, class-imbalance learning and large-scale face recognition, showing consistent improvements in all tasks. Moreover, MHE can significantly improve the image generation quality of GANs (see section D.7). In summary, our paper casts a novel view on regularizing the neurons by introducing hyperspherical diversity.

CHAPTER 6

MINIMIZING COMPRESSIVE HYPERSPHERICAL ENERGY

6.1 Introduction

Recent years have witnessed the tremendous success of deep neural networks in a variety of tasks. With its over-parameterization nature and hierarchical structure, deep neural networks achieve unprecedented performance on many challenging problems [2, 78, 3], but their strong approximation ability also makes it easy to overfit the training set, which greatly affects the generalization on unseen samples. Therefore, how to restrict the huge parameter space and properly regularize the deep networks becomes increasingly important. Regularizations for neural networks can be roughly categorized into *implicit* and *explicit* ones. Implicit regularizations usually do not directly impose explicit constraints on neuron weights, and instead they regularize the networks in an implicit manner in order to prevent overfitting and stabilize the training. A lot of prevailing methods fall into this category, such as batch normalization [44], dropout [116], weight normalization [128], etc. Explicit regularizations [129, 30, 29, 130, 131, 31] usually introduce some penalty terms for neuron weights, and jointly optimize them along with the other objective functions.

Among many existing explicit regularizations, minimum hyperspherical energy (MHE) [31] stands out as a simple yet effective regularization that promotes the *hyperspherical diversity* among neurons and significantly improves the network generalization. MHE regularizes the directions of neuron weights by minimizing a potential energy on a unit hypersphere that characterizes the hyperspherical diversity (such energy is defined as *hyperspherical energy* [31]). In contrast, standard weight decay only regularizes the norm of neuron weights, which essentially can be viewed as regularizing one dimension of the weights. MHE completes an important missing piece by regularizing the neuron directions

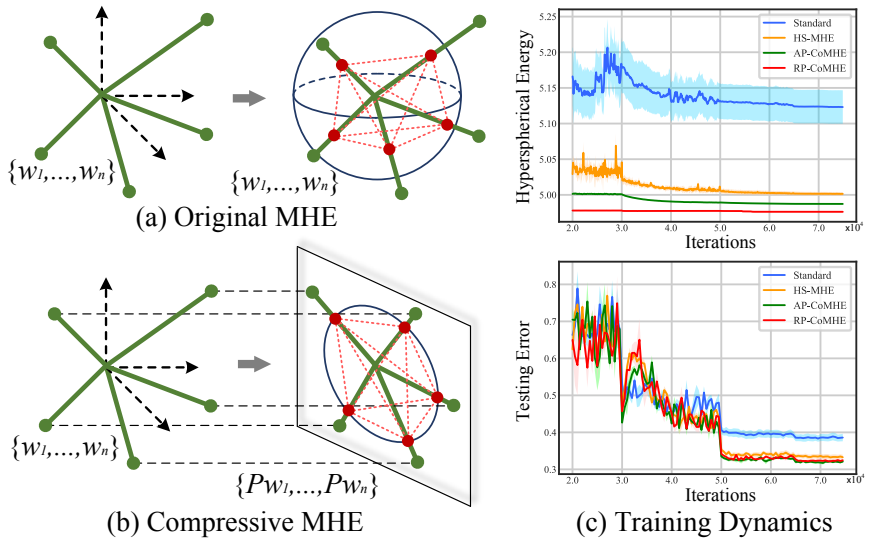


Figure 6.1: Comparison of original MHE and compressive MHE. In (c), the top figure shows the hyperspherical energy, and the bottom one shows the testing error (CIFAR-100). Experimental details are given in section E.2.

(*i.e.*, regularizing the rest dimensions of the weights).

Although minimizing hyperspherical energy has already been empirically shown useful in a number of applications [31], two fundamental questions remain unanswered: *(1) what is the role that hyperspherical energy plays in training a well-performing neural network?* and *(2) How can the hyperspherical energy be effectively minimized?* To study the first question, we plot the training dynamics of hyperspherical energy (on CIFAR-100) in Figure 6.1(c) for a baseline convolutional neural network (CNN) without any MHE variant, a CNN regularized by MHE [31] and a CNN regularized by our CoMHE. More experimental details and full results (with more interesting baselines) are given in section E.2. From the empirical results in Figure 6.1(c), we find that both MHE and CoMHE can achieve much lower hyperspherical energy and testing error than the baseline, showing the effectiveness of minimizing hyperspherical energy. It also implies that lower hyperspherical energy typically leads to better generalization. We empirically observe that a trained neural network with lower hyperspherical energy often generalizes better (*i.e.*, higher hyperspherical diversity leads to better generalization), and therefore we argue that hyperspherical energy is closely related to the generalization power of neural networks. In the rest of the paper, we

delve into the second question that remains an open challenge: how to effectively minimize hyperspherical energy.

By adopting the definition of hyperspherical energy as the regularization objective and naively minimizing it with back-propagation, MHE suffers from a few critical problems which limit it to further unleash its potential. First, the original MHE objective has a huge number of local minima and stationary points due to its highly non-convex and non-linear objective function. The problem can get even worse when the space dimension gets higher and the number of neurons becomes larger [132, 133]. Second, the gradient of the original MHE objective *w.r.t* the neuron weight is deterministic. Unlike the weight decay whose objective is convex, MHE has a complex and non-convex regularization term. Therefore, deterministic gradients may make the solution quickly fall into one of the bad local minima and get stuck there. Third, MHE defines an ill-posed problem in general. When the number of neurons is smaller than the dimension of the space (it is often the case in neural networks), it will be less meaningful to encourage the hyperspherical diversity since the neurons can not fully occupy the space. Last, in high-dimensional spaces, randomly initialized neurons are likely to be orthogonal to each other (see section E.3). Therefore, these high-dimensional neurons can be trivially "diverse", leading to small gradients in original MHE that cause optimization difficulties.

In order to address these problems and effectively minimize hyperspherical energy, we propose the compressive minimum hyperspherical energy (CoMHE) as a generic regularization for neural networks. The high-level intuition behind CoMHE is to project neurons to some suitable subspaces such that the hyperspherical energy can get minimized more effectively. Specifically, CoMHE first maps the neurons from a high-dimensional space to a low-dimensional one and then minimizes the hyperspherical energy of these neurons. Therefore, how to map these neurons to a low-dimensional space while preserving the desirable information in high-dimensional space is our major concern. Since we aim to regularize the directions of neurons, what we care most is the angular similarity between

different neurons. To this end, we explore multiple novel methods to perform the projection and heavily study two main approaches: *random projection* and *angle-preserving projection*, which can reduce the dimensionality of neurons while still partially preserving the pairwise angles.

Random projection (RP) is a natural choice to perform the dimensionality reduction in MHE due to its simplicity and nice theoretical properties. RP can provably preserve the angular information, and most importantly, introduce certain degree of randomness to the gradients, which may help CoMHE escape from some bad local minima. The role that the randomness serves in CoMHE is actually similar to the *simulated annealing* [134, 135] that is widely used to solve Thomson problem. Such randomness is often shown to benefit the generalization [136, 137]. We also provably show that using RP can well preserve the pairwise angles between neurons. Besides RP, we propose the angle-preserving projection (AP) as an effective alternative. AP is motivated by the goal that we aim to preserve the pairwise angles between neurons. Constructing an AP that can project neurons to a low-dimensional space that well preserves the angles is often difficult even with powerful non-linear functions, which is suggested by the strong conditions required for conformal mapping in complex analysis [138]. Therefore, we frame the AP construction as an optimization problem which can be solved jointly with hyperspherical energy minimization. More interestingly, we consider the *adversarial projection* for CoMHE, which minimizes the maximal energy attained by learning the projection. We formulate it as a min-max optimization and optimize it jointly with the neural network.

However, it is inevitable to lose some information in low-dimensional spaces and the neurons may only get diverse in some particular low-dimensional spaces. To address it, we adopt multiple projections to better approximate the MHE objective in the original high-dimensional space. Specifically, we project the neurons to multiple subspaces, compute the hyperspherical energy in each space separately and then minimize the aggregation (*i.e.*, average or max). Moreover, we reinitialize these projection matrix randomly every certain

number of iterations to avoid trivial solutions.

In contrast to MHE that imposes a static regularization to the neurons, CoMHE dynamically regularizes the neurons based on the projection matrices. Such dynamic regularization is equivalent to adjusting the CoMHE objective function, making it easier to escape some bad local minima. Our contributions can be summarized as:

- We first show that hyperspherical energy is closely related to generalization and then reveal the role it plays in training a neural network that generalizes well.
- To address the drawbacks of MHE, we propose CoMHE as a dynamic regularization to effectively minimize hyperspherical energy of neurons for better generalizability.
- We explore different ways to construct a suitable projection for CoMHE. Random projection and angle-preserving projection are proposed to reduce the dimensionality of neurons while preserving the angular information. We also consider several variants such as adversarial projection CoMHE and group CoMHE.
- We provide some theoretical insights for the proposed projections on the quality of preserving the angular similarity between different neurons.
- We show that CoMHE consistently outperforms the original MHE in different tasks. Notably, a 9-layer plain CNN regularized by CoMHE outperforms a standard 1001-layer ResNet by more than 2% on CIFAR-100.

6.2 Related Work

Diversity-based regularization has been found useful in sparse coding [110, 111], ensemble learning [112, 113], self-paced learning [114], metric learning [115], latent variable models [107], etc. Early studies in sparse coding [110, 111] model the diversity with the empirical covariance matrix and show that encouraging such diversity can improve the dictionary's generalizability. [105] promotes the uniformity among eigenvalues of the component matrix in a latent space model. [26, 27, 28, 29, 30, 28] characterize diversity among neurons with orthogonality, and regularize the neural network by promoting the orthogo-

nality. Inspired by the Thomson problem in physics, MHE [31] defines the hyperspherical energy to characterize the diversity on a unit hypersphere and shows significant and consistent improvement in supervised learning tasks. There are two MHE variants in [31]: full-space MHE and half-space MHE. Compared to full-space MHE, the half-space variant [31] further eliminates the collinear redundancy by constructing virtual neurons with the opposite direction to the original ones and then minimizing their hyperspherical energy together. The importance of regularizing angular information is also discussed in [9, 12, 10, 20, 15, 13, 11, 14, 33, 22].

6.3 Compressive MHE

6.3.1 Revisiting Standard MHE

MHE characterizes the diversity of N neurons ($\boldsymbol{W}_N = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_N \in \mathbb{R}^{d+1}\}$) on a unit hypersphere using hyperspherical energy which is defined as

$$\begin{aligned} \boldsymbol{E}_{s,d}(\hat{\boldsymbol{w}}_i|_{i=1}^N) &= \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} f_s\big(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|\big) \\ &= \begin{cases} \sum_{i\neq j} \|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-s}, & s > 0 \\ \sum_{i\neq j} \log\big(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-1}\big), & s = 0 \end{cases} \end{aligned} \tag{6.1}$$

where $\|\cdot\|$ denotes ℓ_2 norm, $f_s(\cdot)$ is a decreasing real-valued function (we use $f_s(z) = z^{-s}, s > 0$, *i.e.*, Riesz s-kernels), and $\hat{\boldsymbol{w}}_i = \frac{\boldsymbol{w}_i}{\|\boldsymbol{w}_i\|}$ is the i-th neuron weight projected onto the unit hypersphere $\mathbb{S}^d = \{\boldsymbol{v} \in \mathbb{R}^{d+1} | \, \|\boldsymbol{v}\| = 1\}$. For convenience, we denote $\hat{\boldsymbol{W}}_N = \{\hat{\boldsymbol{w}}_1, \cdots, \hat{\boldsymbol{w}}_N \in \mathbb{S}^d\}$, and $\boldsymbol{E}_s = \boldsymbol{E}_{s,d}(\hat{\boldsymbol{w}}_i|_{i=1}^N)$. Note that, each neuron is a convolution kernel in CNNs. MHE minimizes the hyperspherical energy of neurons using gradient descent during back-propagation, and MHE is typically applied to the neural network in a layer-wise fashion. We first write down the gradient of $\boldsymbol{E}_2$ *w.r.t* $\hat{\boldsymbol{w}}_i$ and make the gradient

to be zero:

$$\nabla_{\hat{\boldsymbol{w}}_i} \boldsymbol{E}_2 = \sum_{j=1,j\neq i}^{N} \frac{-2(\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j)}{\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^4} = 0 \Rightarrow \hat{\boldsymbol{w}}_i = \frac{\sum_{j=1,j\neq i}^{N} \alpha_j \hat{\boldsymbol{w}}_j}{\sum_{j=1,j\neq i}^{N} \alpha_j} \tag{6.2}$$

where $\alpha_j = \|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-4}$. We use toy and informal examples to show that high dimensional space (*i.e.*, d is large) leads to much more stationary points than low-dimensional one. Assume there are $K = K_1 + K_2$ stationary points in total for $\hat{\boldsymbol{W}}_N$ to satisfy Equation 6.2, where K_1 denotes the number of stationary points in which every element in the solution is distinct and K_2 denotes the number of the rest stationary points. We give two examples: *(i)* For $(d+2)$-dimensional space, we can extend the solutions in $(d+1)$-dimensional space by introducing a new dimension with zero value. The new solutions satisfy Equation 6.2. Because there are $d+2$ ways to insert the zero, we have at least $(d+2)K$ stationary points in $(d+2)$-dimensional space. *(ii)* We denote $K_1' = \frac{K_1}{(d+1)!}$ as the number of unordered sets that construct the stationary points. In $(2d+2)$-dimensional space, we can construct $\hat{\boldsymbol{w}}_j^E = \frac{1}{\sqrt{2}}\{\hat{\boldsymbol{w}}_j; \hat{\boldsymbol{w}}_j\} \in \mathbb{S}^{2d+1}, \forall j$ that satisfies Equation 6.2. Therefore, there are at least $\frac{(2d+2)!}{2^{d+1}} K_1' + K_2$ stationary points for $\hat{\boldsymbol{W}}_N$ in $(2d+2)$-dimensional space, and besides this construction, there are much more stationary points. Therefore, MHE have far more stationary points in higher dimensions.

6.3.2 General Framework

To overcome MHE's drawbacks in high dimensional space, we propose the compressive MHE that projects the neurons to a low-dimensional space and then minimizes the hyperspherical energy of the projected neurons. In general, CoMHE minimizes the following form of energy:

$$\boldsymbol{E}_s^C(\hat{\boldsymbol{W}}_N) := \sum_{i=1}^{N} \sum_{j=1,j\neq i}^{N} f_s\big(\|g(\hat{\boldsymbol{w}}_i) - g(\hat{\boldsymbol{w}}_j)\|\big) \tag{6.3}$$

where $g : \mathbb{S}^d \to \mathbb{S}^k$ takes a normalized $(d+1)$-dimensional input and outputs a normalized $(k+1)$-dimensional vector. $g(\cdot)$ can be either linear or nonlinear mapping. We only consider the linear case here. Using multi-layer perceptrons as $g(\cdot)$ is one of the simplest nonlinear cases. Similar to MHE, CoMHE also serves as a regularization in neural networks.

6.3.3 Random Projection for CoMHE

Random projection is in fact one of the most straightforward way to reduce dimensionality while partially preserving the angular information. More specifically, we use a random mapping $g(\boldsymbol{v}) = \frac{\boldsymbol{P}\boldsymbol{v}}{\|\boldsymbol{P}\boldsymbol{v}\|}$ where $\boldsymbol{P} \in \mathbb{R}^{(k+1)\times(d+1)}$ is a Gaussian distributed random matrix (each entry follows i.i.d. normal distribution). In order to reduce the variance, we use C random projection matrices to project the neurons and compute the hyperspherical energy separately:

$$\boldsymbol{E}_s^R(\hat{\boldsymbol{W}}_N) := \frac{1}{C}\sum_{c=1}^{C}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s\Big(\Big\|\frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_i}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_i\|} - \frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_j}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_j\|}\Big\|\Big) \tag{6.4}$$

where $\boldsymbol{P}_c, \forall c$ is a random matrix with each entry following the normal distribution $\mathcal{N}(0,1)$. According to the properties of normal distribution [139], every normalized row of the random matrix $\boldsymbol{P}$ is uniformly distributed on a hypersphere $\mathbb{S}^d$, which indicates that the projection matrix $\boldsymbol{P}$ is able to cover all the possible subspaces. Multiple projection matrices can also be interpreted as multi-view projection, because we are making use of information from multiple projection views. In fact, we do not necessarily need to average the energy for multiple projections, and instead we can use maximum operation (or some other meaningful aggregation operations). Then the objective becomes

$$\max_c \sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s(\|\frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_i}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_i\|} - \frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_j}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_j\|}\|). \tag{6.5}$$

Considering that we aim to minimize this objective, the problem is in fact a min-max optimization. Note that, we will typically re-initialize the random projection matrices every certain number of iterations to avoid trivial solutions. Most importantly, using RP can provably preserve the angular similarity.

6.3.4 Angle-preserving Projection for CoMHE

Recall that we aim to find a projection to project the neurons to a low-dimensional space that best preserves angular information. We transform the goal to an optimization:

$$\boldsymbol{P}^\star = \arg\min_{\boldsymbol{P}} \mathcal{L}_P := \sum_{i \neq j} (\theta_{(\hat{\boldsymbol{w}}_i, \hat{\boldsymbol{w}}_j)} - \theta_{(\boldsymbol{P}\hat{\boldsymbol{w}}_i, \boldsymbol{P}\hat{\boldsymbol{w}}_j)})^2 \tag{6.6}$$

where $\boldsymbol{P} \in \mathbb{R}^{(k+1)\times(d+1)}$ is the projection matrix and $\theta_{(\boldsymbol{v}_1, \boldsymbol{v}_2)}$ denotes the angle between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. For implementation convenience, we can replace the angle with the cosine value (*e.g.*., use $\cos(\theta_{(\hat{\boldsymbol{w}}_i, \hat{\boldsymbol{w}}_j)})$ to replace $\theta_{(\hat{\boldsymbol{w}}_i, \hat{\boldsymbol{w}}_j)}$), so that we can directly use the inner product of normalized vectors to measure the angular similarity. With $\hat{\boldsymbol{P}}$ obtained in Equation 6.6, we use a nested loss function:

$$\begin{aligned} \boldsymbol{E}_s^A(\hat{\boldsymbol{W}}_N, \boldsymbol{P}^\star) := \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} f_s\Big(\Big\| \frac{\boldsymbol{P}^\star \hat{\boldsymbol{w}}_i}{\|\boldsymbol{P}^\star \hat{\boldsymbol{w}}_i\|} - \frac{\boldsymbol{P}^\star \hat{\boldsymbol{w}}_j}{\|\boldsymbol{P}^\star \hat{\boldsymbol{w}}_j\|} \Big\| \Big) \\ \text{s.t.} \quad \boldsymbol{P}^\star = \arg\min_{\boldsymbol{P}} \sum_{i \neq j} (\theta_{(\hat{\boldsymbol{w}}_i, \hat{\boldsymbol{w}}_j)} - \theta_{(\boldsymbol{P}\hat{\boldsymbol{w}}_i, \boldsymbol{P}\hat{\boldsymbol{w}}_j)})^2 \end{aligned} \tag{6.7}$$

for which we propose two different ways to optimize the projection matrix $\boldsymbol{P}$. We can approximate $\boldsymbol{P}^\star$ using a few gradient descent updates. Specifically, we use two different ways to perform the optimization. Naively, we use a few gradient descent steps to update $\boldsymbol{P}$ in order to approximate $\boldsymbol{P}^\star$ and then update $\boldsymbol{W}_N$, which proceeds alternately. The number of iteration steps that we use to update $\boldsymbol{P}$ is a hyperparemter and needs to be determined by cross-validation. Besides the naive alternate one, we also use a different optimization of $\boldsymbol{W}_N$ by unrolling the gradient update of $\boldsymbol{P}$.

Alternating optimization. The alternating optimization is to optimize $\boldsymbol{P}$ alternately with the network parameters $\boldsymbol{W}_N$. Specifically, in each iteration of updating the network parameters, we update $\boldsymbol{P}$ every number of inner iterations and use it as an approximation to $\boldsymbol{P}^\star$ (the error depends on the number of gradient steps we take). Essentially, we are alternately solving two separate optimization problems for $\boldsymbol{P}$ and $\boldsymbol{W}_N$ with gradient descent.

Unrolled optimization. Instead of naively updating $\boldsymbol{W}_N$ with approximate $\boldsymbol{P}^\star$ in the alternating optimization, the unrolled optimization further unrolls the update rule of $\boldsymbol{P}$ and embed it within the optimization of network parameters $\boldsymbol{W}_N$. If we denote the CoMHE loss with a given projection matrix $\boldsymbol{P}$ as $\boldsymbol{E}_s^A(\boldsymbol{W}_N, \boldsymbol{P})$ which takes $\boldsymbol{W}_N$ and $\boldsymbol{P}$ as input, then the unrolled optimization is essentially optimizing $\boldsymbol{E}_s^A(\boldsymbol{W}_N, \boldsymbol{P} - \eta \cdot \frac{\partial \mathcal{L}_P}{\partial \boldsymbol{P}})$. It can also be viewed as minimizing the CoMHE loss after a single step of gradient descent *w.r.t.* the projection matrix. This optimization includes the computation of second-order partial derivatives. Note that, it is also possible to unroll multiple gradient descent steps. Similar unrolling is also applied in [140, 141, 142].

6.3.5 Notable CoMHE Variants

We provide more interesting CoMHE variants as an extension. We will have some preliminary empirical study on these variants, but our main focus is still on RP and AP.

Adversarial Projection for CoMHE. We consider a novel CoMHE variant that adversarially learns the projection. The intuition behind is that we want to learn a projection basis that maximizes the hyperspherical energy while the final goal is to minimize this maximal energy. With such intuition, we can construct a min-max optimization:

$$\min_{\hat{\boldsymbol{W}}_N} \max_{\boldsymbol{P}} \boldsymbol{E}_s^V(\hat{\boldsymbol{W}}_N, \boldsymbol{P}) := \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} f_s\Big(\Big\| \frac{\boldsymbol{P}\hat{\boldsymbol{w}}_i}{\|\boldsymbol{P}\hat{\boldsymbol{w}}_i\|} - \frac{\boldsymbol{P}\hat{\boldsymbol{w}}_j}{\|\boldsymbol{P}\hat{\boldsymbol{w}}_j\|} \Big\| \Big) \tag{6.8}$$

which can be solved by gradient descent similar to [143]. From a game-theoretical per-

spective, $\boldsymbol{P}$ and $\hat{\boldsymbol{W}}_N$ can be viewed as two players that are competing with each other. However, due to the instability of solving the min-max problem, the performance of this projection is unstable.

Group CoMHE. Group CoMHE is a very special case in the CoMHE framework. The basic idea is to divide the weights of each neuron into several groups and then minimize the hyperspherical energy within each group. For example in CNNs, group MHE divides the channels into groups and minimizes within each group the MHE loss. Specifically, the objective function of group CoMHE is

$$\boldsymbol{E}_s^G(\hat{\boldsymbol{W}}_N) := \frac{1}{C}\sum_{c=1}^{C}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s\Big(\Big\|\frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_i}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_i\|} - \frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_j}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_j\|}\Big\|\Big) \tag{6.9}$$

where $\boldsymbol{P}_c$ is a diagonal matrix with every diagonal entry being either 0 or 1, and $\sum_c \boldsymbol{P}_c = \boldsymbol{I}$ (in fact, this is optional). There are multiple ways to divide groups for the neurons, and typically we will divide groups according to the channels, similar to [144]. More interestingly, one can also divide the groups in a *stochastic* fashion.

6.3.6 Shared Projection Basis in Neural Networks

In general, we usually need different projection bases for neurons in different layers of the neural network. However, we find it beneficial to share some projection bases across different layers. We only share the projection matrix for the neurons in different layers that have the same dimensionality. For example in a neural network, if the neurons in the first layer have the same dimensionality with the neurons in the second layer, we will share their projection matrix that reduces the dimensionality. Sharing the projection basis can effectively reduce the number of projection parameters and may also reduce the inconsistency within the hyperspherical energy minimization of projected neurons in different layers. Most importantly, it can empirically improve the network generalizability while using much fewer parameters and saving more computational overheads.

6.4 Theoretical Insights

6.4.1 Angle Preservation

We start with highly relevant properties of random projection and then delve into the angular preservation.

Lemma 3 (Mean Preservation of Random Projection)**.** *For any $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^d$ and any random Gaussian distributed matrix $\boldsymbol{P} \in \mathbb{R}^{k\times d}$ where $\boldsymbol{P}_{ij} = \frac{1}{\sqrt{n}} r_{ij}$, if $r_{ij}, \forall i, j$ are i.i.d. random variables from $\mathcal{N}(0, 1)$, we have $\mathbb{E}(\langle \boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2\rangle) = \langle \boldsymbol{w}_1, \boldsymbol{w}_2\rangle$.*

This lemma indicates that the mean of randomly projected inner product is well preserved, partially justifying why using random projection actually makes senses.

Johnson-Lindenstrauss lemma (JLL) [145, 146] (in section E.4) establishes a guarantee for the Euclidean distance between randomly projected vectors. However, JLL does not provide the angle preservation guarantees. It is nontrivial to provide a guarantee for angular similarity from JLL.

Theorem 3 (Angle Preservation I)**.** *Given $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^d$, $\boldsymbol{P} \in \mathbb{R}^{k\times d}$ is a random projection matrix that has i.i.d. 0-mean σ-subgaussian entries, and $\boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2 \in \mathbb{R}^k$ are the randomly projected vectors of $\boldsymbol{w}_1, \boldsymbol{w}_2$ under $\boldsymbol{P}$. Then $\forall \epsilon \in (0, 1)$, we have that*

$$\frac{\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}) - \epsilon}{1+\epsilon} < \cos(\theta_{(\boldsymbol{P}\boldsymbol{w}_1,\boldsymbol{P}\boldsymbol{w}_2)}) < \frac{\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}) + \epsilon}{1-\epsilon} \tag{6.10}$$

which holds with probability $\left(1 - 2\exp(-\frac{k\epsilon^2}{8})\right)^2$.

Theorem 4 (Angle Preservation II)**.** *Given $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^d$, $\boldsymbol{P} \in \mathbb{R}^{k\times d}$ is a Gaussian random projection matrix where $\boldsymbol{P}_{ij} = \frac{1}{\sqrt{n}} r_{ij}$ ($r_{ij}, \forall i, j$ are i.i.d. random variables from $\mathcal{N}(0, 1)$), and $\boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2 \in \mathbb{R}^k$ are the randomly projected vectors of $\boldsymbol{w}_1, \boldsymbol{w}_2$ under $\boldsymbol{P}$. Then $\forall \epsilon \in$*

$(0,1)$ *and* $\boldsymbol{w}_1^\top \boldsymbol{w}_2 > 0$*, we have that*

$$\begin{aligned} \frac{1+\epsilon}{1-\epsilon}\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}) - \frac{2\epsilon}{1-\epsilon} &< \cos(\theta_{(\boldsymbol{P}\boldsymbol{w}_1,\boldsymbol{P}\boldsymbol{w}_2)}) \\ &< \frac{1-\epsilon}{1+\epsilon}\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}) + \frac{1+2\epsilon}{1+\epsilon} - \frac{\sqrt{(1-\epsilon^2)}}{1+\epsilon} \end{aligned} \tag{6.11}$$

which holds with probability $1 - 6\exp(-\frac{k}{2}(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3}))$.

Theorem Theorem 3 is one of our main theoretical results and reveals that the angle between randomly projected vectors is well preserved. Note that, the parameter σ of the subgaussian distribution is not related to our bound for the angle, so any Gaussian distributed random matrix has the property of angle preservation. The projection dimension k is related to the probability that the angle preservation bound holds. Theorem Theorem 4 is a direct result from [147]. It again shows that the angle between randomly projected vectors is provably preserved. Both Theorem Theorem 3 and Theorem Theorem 4 give upper and lower bounds for the angle between randomly projected vectors. If $\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)} > \arccos(\frac{\epsilon+3\epsilon^2}{3\epsilon+\epsilon^2})$, then the lower bound in Theorem Theorem 3 is tighter than the lower bound in Theorem Theorem 4. If $\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)} > \arccos(\frac{1-3\epsilon^2-(1-\epsilon)\sqrt{1-\epsilon^2}}{3\epsilon-\epsilon^2})$, the upper bound in Theorem Theorem 3 is tighter than the upper bound in Theorem Theorem 4. To conclude, Theorem Theorem 3 gives tighter bounds when the angle of original vectors is large. Since AP is randomly initialized every certain number of iterations and minimizes the angular difference before and after the projection, AP usually performs better than RP in preserving angles. Without the angle-preserving optimization, AP reduces to RP.

6.4.2 Statistical Insights

We can also draw some theoretical intuitions from spherical uniform testing [148] in statistics. Spherical uniform testing is a nonparametric statistical hypothesis test that checks whether a set of observed data is generated from a uniform distribution on a hypersphere or not. Random projection is in fact an important tool [148] in statistics to test the uniformity

on hyperspheres, while our goal is to promote the same type of hyperspherical uniformity (*i.e.*, diversity). Specifically, we have N random samples $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_N$ of $\mathbb{S}^d$-valued random variables, and the random projection $\boldsymbol{p}$ which is another random variable independent of $\boldsymbol{w}_i, \forall i$ and uniformly distributed on $\mathbb{S}^d$. The projected points of $\boldsymbol{w}_i, \forall i$ is $y_i = \boldsymbol{p}^\top \boldsymbol{w}_i, \forall i$. The distribution of $y_i, \forall i$ uniquely determines the distribution of $\boldsymbol{w}_1$, as is specified by Theorem Theorem 5.

Theorem 5 (Unique Distribution Determination of Random Projection)**.** *Let $\boldsymbol{w}$ be a $\mathbb{S}^d$-valued random variable and $\boldsymbol{p}$ be a random variable that is uniformly distributed on $\mathbb{S}^d$ and independent of $\boldsymbol{w}$. With probability one, the distribution of $\boldsymbol{w}$ is uniquely determined by the distribution of the projection of $\boldsymbol{w}$ on $\boldsymbol{p}$. More specifically, if $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are $\mathbb{S}^d$-valued random variables, independent of $\boldsymbol{p}$ and we have a positive probability for the event that $\boldsymbol{p}$ takes a value $\boldsymbol{p}_0$ such that the two distributions satisfy $\boldsymbol{p}_0^\top \boldsymbol{w}_1 \sim \boldsymbol{p}_0^\top \boldsymbol{w}_2$, then $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are identically distributed.*

Theorem Theorem 5 shows that the distributional information is well preserved after random projection, providing the CoMHE framework a statistical intuition and foundation. We emphasize that the randomness here is in fact very crucial. For a fixed projection $\boldsymbol{p}_0$, Theorem Theorem 5 does not hold in general. As a result, random projection for CoMHE is well motivated from the statistical perspective.

6.4.3 Insights from Random Matrix Theory

Random projection may also impose some implicit regularization to learning the neuron weights. [149] proves that random projection serves as a regularizer for the Fisher linear discrimination classifier. From metric learning perspective, the inner product between neurons $\boldsymbol{w}_1^\top \boldsymbol{w}_2$ will become $\boldsymbol{w}_1^\top \boldsymbol{P}^\top \boldsymbol{P} \boldsymbol{w}_2$ where $\boldsymbol{P}^\top \boldsymbol{P}$ defines a specific form of (low-rank) similarity [150, 33]. [151] proves that random projection satisfying the JLL *w.h.p* also satisfies the restricted isometry property (RIP) *w.h.p* under sparsity assumptions. In this case,

the neuron weights can be well recovered [152, 153]. These results imply that randomly projected neurons in CoMHE may implicitly regularize the network.

6.5 Discussions and Extensions

Bilateral projection for CoMHE. If we view the neurons in one layer as a matrix $\boldsymbol{W} = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_n\} \in \mathbb{R}^{m \times n}$ where m is the dimension of neurons and n is the number of neurons, then the projection considered throughout the paper is to left-multiply a projection matrix $\boldsymbol{P}_1 \in \mathbb{R}^{r \times m}$ to $\boldsymbol{W}$. In fact, we can further reduce the number of neurons by right-multiplying an additional projection matrix $\boldsymbol{P}_2 \in \mathbb{R}^{n \times r}$ to $\boldsymbol{W}$. Specifically, we denote that $\boldsymbol{Y}_1 = \boldsymbol{P}_1\boldsymbol{W}$ and $\boldsymbol{Y}_2 = \boldsymbol{W}\boldsymbol{P}_2$. Then we can apply the MHE regularization separately to column vectors of $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$. The final neurons are still $\boldsymbol{W}$. More interestingly, we can also approximate $\boldsymbol{W}$ with a low-rank factorization [154]: $\tilde{\boldsymbol{W}} = \boldsymbol{Y}_2(\boldsymbol{P}_1\boldsymbol{Y}_2)^{-1}\boldsymbol{Y}_1 \in \mathbb{R}^{m \times n}$. It inspires us to directly use two set of parameters $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ to represent the equivalent neurons $\tilde{\boldsymbol{W}}$ and apply the MHE regularization separately to their column vectors. Different from the former case, we use $\tilde{\boldsymbol{W}}$ as the final neurons. More details are in section E.6.

Constructing random projection matrices. In random projection, we typically construct random matrices with each element drawn *i.i.d.* from a normal distribution. However, there are many more choices for constructing a random matrices that can provably preserve distance information. For example, we have subsampled randomized Hadamard transform [155] and count sketch-based projections [156].

Comparison to existing works. One of the widely used regularizations is the orthonormal regularization [12, 157] that minimizes $\|\boldsymbol{W}^\top\boldsymbol{W} - \boldsymbol{I}\|_F$ where W denotes the weights of a group of neurons with each column being one neuron and $\boldsymbol{I}$ is an identity matrix. [29, 27] are also built upon orthogonality. In contrast, both MHE and CoMHE do not encourage orthogonality among neurons and instead promote hyperspherical uniformity and diversity.

Randomness improves generalization. Both RP and AP introduce randomness to CoMHE, and the empirical results show that such randomness can greatly benefit the net-

Table 6.1: CoMHE variants on CIFAR-100.

Method	Error (%)
Baseline	28.03
Orthogonal	27.01
SRIP [29]	25.80
MHE [31]	26.75
HS-MHE [31]	25.96
G-CoMHE	25.08
Adv-CoMHE	25.09
RP-CoMHE	24.39
RP-CoMHE (max)	24.77
AP-CoMHE (alter.)	24.95
AP-CoMHE (unroll)	**24.33**

work generalization. It is well-known that stochastic gradient is one of the key ingredients that help neural networks generalize well to unseen samples. Interestingly, randomness in CoMHE also leads to a stochastic gradient. [136] also theoretically shows that randomness helps generalization, partially justifying the effectiveness of CoMHE.

6.6 Experiments and Results

6.6.1 Image Recognition

We perform image recognition to show the improvement of regularizing CNNs with CoMHE. Our goal is to show the superiority of CoMHE rather than achieving state-of-the-art accuracies on particular tasks. For all the experiments on CIFAR-10 and CIFAR-100 in the paper, we use the same data augmentation as [2, 20]. For ImageNet-2012, we use the same data augmentation in [12]. We train all the networks using SGD with momentum 0.9. All the networks use BN [44] and ReLU if not otherwise specified. By default, all CoMHE variants are built upon half-space MHE. Experimental details are given in each subsection and section E.1. More experiments are given in section E.9, section E.8 and section E.10.

Table 6.2: Error (%) on CIFAR-100 under different dimension of projection.

Projection Dimension	10	20	30	40	80
RP-CoMHE	25.48	25.32	24.60	**24.75**	25.46
AP-CoMHE (alter.)	**25.21**	24.60	24.95	24.97	**24.99**
AP-CoMHE (unroll.)	25.32	**24.59**	**24.33**	24.93	25.12

Ablation Study and Exploratory Experiments

Variants of CoMHE. We compare different variants of CoMHE with the same plain CNN-9 (section E.1). Specifically, we evaluate the baseline CNN without any regularization, half-space MHE (HS-MHE) which is the best MHE variant from [31], random projection CoMHE (RP-CoMHE), RP-CoMHE (max) that uses max instead of average for loss aggregation, angle-preserving projection CoMHE (AP-CoMHE), adversarial projection CoMHE (Adv-CoMHE) and group CoMHE (G-CoMHE) on CIFAR-100. For RP, we set the projection dimension to 30 (*i.e.*, $k = 29$) and the number of projection to 5 (*i.e.*, $C = 5$). For AP, the number of projection is 1 and the projection dimension is set to 30. For AP, we evaluate both alternating optimization and unrolled optimization. In alternating optimization, we update the projection matrix every 10 steps of network update. In unrolled optimization, we only unroll one-step gradient in the optimization. For G-CoMHE, we construct a group with every 8 consecutive channels. All these design choices are obtained using cross-validation. We will also study how these hyperparameters affect the performance in the following experiments. The results in Table 6.1 show that all of our proposed CoMHE variants can outperform the original half-space MHE by a large margin. The unrolled optimization in AP-CoMHE shows the significant advantage over alternating one and achieves the best accuracy. Both Adv-CoMHE and G-CoMHE achieve decent performance gain over HS-MHE, but not as good as RP-CoMHE and AP-CoMHE. Therefore, we will mostly focus on RP-CoMHE and AP-CoMHE in the remaining experiments.

Dimension of projection. We evaluate how the dimension of projection (*i.e.*, k) affects the performance. We use the plain CNN-9 as the backbone network and test on CIFAR-100. We fix the number of projections in RP-CoMHE to 20. Because AP-CoMHE does not need

Table 6.3: Error (%) on CIFAR-100 under different numbers of projections.

# Proj.	RP-CoMHE	AP-CoMHE
1	25.11	**24.33**
5	**24.39**	24.34
10	25.11	24.36
20	24.60	24.38
30	24.82	24.52
80	24.92	24.56

Table 6.4: Error (%) on CIFAR-100 with different network width.

Width	$t = 1$	$t = 2$	$t = 4$	$t = 8$	$t = 16$	$t = 20$
Baseline	47.72	38.64	28.13	24.95	24.44	23.77
MHE [31]	36.84	30.05	26.75	24.05	23.14	22.36
HS-MHE [31]	35.16	29.33	25.96	23.38	21.83	21.22
RP-CoMHE	**34.73**	**28.92**	24.39	**22.44**	20.81	20.62
AP-CoMHE	34.89	29.01	**24.33**	22.6	**20.72**	**20.50**

to use multiple projections to reduce variance, we only use one projection in AP-CoMHE. Results are given in Table 6.2. In general, RP-CoMHE and AP-CoMHE with different projection dimensions can consistently and significantly outperform the half-space MHE, validating the effectiveness and superiority of the proposed CoMHE framework. Specifically, we find that both RP-CoMHE and AP-CoMHE usually achieve the best accuracy when the projection dimension is 20 or 30. Since the unrolled optimization in AP-CoMHE is consistently better than the alternating optimization, we stick to the unrolled optimization for AP-CoMHE in the remaining experiments if not otherwise specified.

Number of projections. We evaluate RP-CoMHE under different numbers of projections. We use the plain CNN-9 as the baseline and test on CIFAR-100. Results in Table 6.3 show that the performance of RP-CoMHE is generally not very sensitive to the number of projections. Surprisingly, we find that it is not necessarily better to use more projections for variance reduction. Our experiment show that using 5 projections can achieve the best accuracy. It may be because large variance can help the solution escape bad local minima in the optimization. Note that, we generally do not use multiple projections in AP-CoMHE, because AP-CoMHE optimizes the projection and variance reduction is unnecessary. Our results do not show performance gain by using multiple projections in AP-CoMHE.

Table 6.5: Error (%) on CIFAR-100 with different network depth. N/C denotes Not Converged.

Depth	CNN-6	CNN-9	CNN-15
Baseline	32.08	28.13	N/C
MHE [31]	28.16	26.75	26.90
HS-MHE [31]	27.56	25.96	25.84
RP-CoMHE	26.73	24.39	**24.21**
AP-CoMHE	**26.55**	**24.33**	24.55

Network width. We evaluate RP-CoMHE and AP-CoMHE with different network width on CIFAR-100. We use the plain CNN-9 as our backbone network architecture, and set its filter number in Conv1.x, Conv2.x and Conv3.x (see section E.1) to $16 \times t$, $32 \times t$ and $64 \times t$, respectively. Specifically, we test the cases where $t = 1, 2, 4, 8, 16$. Taking $t = 2$ as an example, then the filter numbers in Conv1.x, Conv2.x and Conv3.x are 32, 64 and 128, respectively. For RP, we set the projection dimension to 30 and the number of projection to 5. For AP, the number of projection is 1 and the projection dimension is set to 30. The results are shown in Table 6.4. Note that, we use the unrolled optimization in AP-CoMHE. From Table 6.4, one can observe that the performance gains of both RP-CoMHE and AP-CoMHE are very consistent and significant. With wider network, CoMHE also achieves better accuracy. Compared to the strong results of half-space MHE, CoMHE still obtains more than 1% accuracy boost under different network width.

Network depth. We evaluate RP-CoMHE and AP-CoMHE with different network depth on CIFAR-100. We use three plain CNNs with 6, 9 and 15 convolution layers, respectively. For all the networks, we set the filter number in Conv1.x, Conv2.x and Conv3.x to 64, 128 and 256, respectively. Detailed network architectures are given in section E.1. For RP, we set the projection dimension to 30 and the number of projection to 5. For AP, the number of projection is 1 and the projection dimension is set to 30. Table 6.5 shows that both RP-CoMHE and AP-CoMHE can outperform half-space MHE by a considerable margin while regularizing a plain CNN with different depth.

Effectiveness of optimization. To verify that our CoMHE can better minimize the

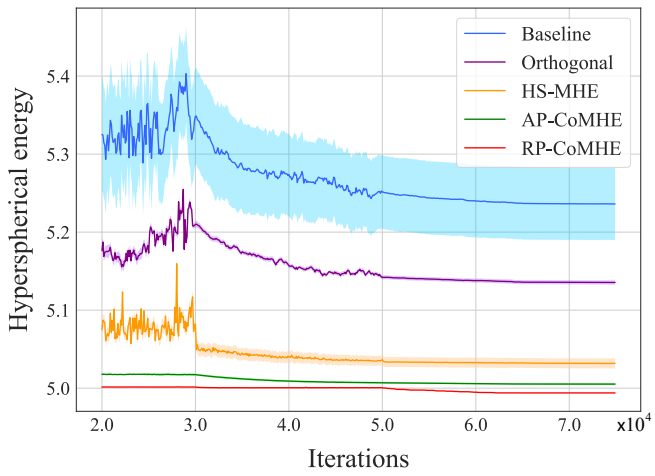


Figure 6.2: Hyperspherical energy during training. All networks are initialized with the same random weights, so the hyperspherical energy is the same before the training starts.

hyperspherical energy, we compute the hyperspherical energy $\boldsymbol{E}_2$ (Equation 6.1) for baseline CNN and CNN regularized by orthogonal regularization, HS-MHE, RP-CoMHE and AP-CoMHE during training. Note that, we compute the original hyperspherical energy rather than the energy after projection. All the networks use exactly the same initialization (the initial hyperspherical energy is the same). The results are averaged over five independent runs. We show the hyperspherical energy after the 20000-th iteration, because at the beginning of the training, hyperspherical energy fluctuates dramatically and is unstable. Interested readers can refer to section E.7 for the complete energy dynamics. From Figure 6.2, one can observe that both RP-CoMHE and AP-CoMHE can better minimize the hyperspherical energy. RP-CoMHE can achieve the lowest energy with smallest standard deviation. From the absolute scale, the optimization gain is also very significant. In the high-dimensional space, the variance of hyperspherical energy is usually small (already close to the smallest energy value) and is already difficult to minimize.

ResNet with CoMHE. All the above experiments are performed using VGG-like plain CNNs, so we use the more powerful ResNet [2] to show that CoMHE is architecture-agnostic. We use the same experimental setting in [85] for fair comparison. We use a standard ResNet-32 as our baseline and the network architecture is specified in section E.1.

Table 6.6: Error (%) using ResNets.

Method	CIFAR-10	CIFAR-100
ResNet-110 [2]	6.61	25.16
ResNet-1001 [85]	4.92	**22.71**
Baseline	5.19	22.87
Orthogonal [27]	5.02	22.36
SRIP [29]	4.75	22.08
MHE [31]	4.72	22.19
HS-MHE [31]	4.66	22.04
RP-CoMHE	4.59	21.82
AP-CoMHE	**4.57**	**21.63**

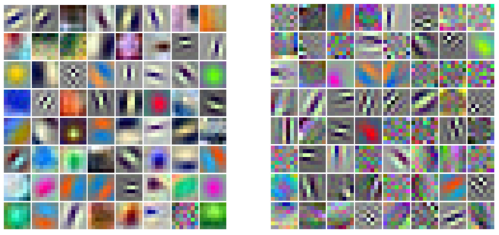
Table 6.7: Top-1 center crop error (%) on ImageNet.

Method	Res-18	Res-34	Res-50
baseline	32.95	30.04	25.30
Orthogonal [27]	32.65	29.74	25.19
Orthnormal [12]	32.61	29.75	25.21
SRIP [29]	32.53	29.55	24.91
MHE [31]	32.50	29.60	25.02
HS-MHE [31]	32.45	29.50	24.98
RP-CoMHE	31.90	29.38	**24.51**
AP-CoMHE	**31.80**	**29.32**	24.53

From the results in Table 6.6, one can observe that both RP-CoMHE and AP-CoMHE can consistently outperform half-space MHE, showing that CoMHE can boost the performance across different network architectures. More interestingly, the ResNet-32 regularized by CoMHE achieves impressive accuracy and is able to outperform the 1001-layer ResNet by a large margin. Additionally, we note that from Table 6.4, we can regularize a plain VGG-like 9-layer CNN with CoMHE and achieve 20.81% error rate, which is nearly 2% improvement over the 1001-layer ResNet.

Large-scale Recognition on ImageNet-2012

We evaluate CoMHE for image recognition on ImageNet-2012 [84]. We perform the experiment using ResNet-18, ResNet-34 and ResNet-50, and then report the top-1 validation error (center crop) in Table 6.7. Our results show consistent and significant performance gain of CoMHE in all ResNet variants. Compared to the baselines, CoMHE can reduce the top-1 error for more than 1%. Since the computational overhead of CoMHE



Baseline CoMHE

Figure 6.3: Visualized first-layer filters.

is almost neglectable, the performance gain is obtained without many efforts. Most importantly, as a plug-in regularization, CoMHE is shown to be architecture-agnostic and produces considerable accuracy gain in most circumstances.

Besides the accuracy improvement, we also visualize in Figure 6.3 the 64 filters in the first-layer learned by the baseline ResNet and the proposed CoMHE-regularized ResNet. The filters look quite different after we regularize the network using CoMHE. Each filter learned by baseline focuses on a particular local pattern (*e.g.*, edge, color and shape) and each one has a clear local semantic meaning. In contrast, filters learned by CoMHE focuses more on edges, textures and global patterns which do not necessarily have a clear local semantic meaning. However, from a representation basis perspective, having such global patterns may be beneficial to the recognition accuracy. We also observe that filters learned by CoMHE pay less attention to color.

6.6.2 Point Cloud Recognition

We evaluate CoMHE on point cloud recognition. Our goal is to validate the effectiveness of CoMHE on a totally different network architecture with a different form of input data structure, rather than achieving state-of-the-art performance on point cloud recognition. To this end, we conduct experiments on widely used neural networks that handles point clouds: PointNet [158] (PN) and PointNet++ [159] (PN++). We combine half-space MHE, RP-CoMHE and AP-CoMHE into PN (without T-Net), PN (with T-Net) and PN++. More experimental details are given in section E.1. We test the performance on ModelNet-

Table 6.8: Accuracy (%) on ModelNet-40.

Method	PN	PN (T)	PN++
Original	87.1	89.20	90.07
MHE [31]	87.31	89.33	90.25
HS-MHE [31]	87.44	89.41	90.31
RP-CoMHE	87.82	89.69	90.52
AP-CoMHE	**87.85**	**89.70**	**90.56**

40 [160]. Specifically, since PN can be viewed as 1×1 convolutions before the max pooling layer, we can apply all these MHE variants similarly to CNN. After the max pooling layer, there is a standard fully connected network where we can still apply the MHE variants. We compare the performance of regularizing PN and PN++ with half-space MHE, RP-CoMHE or AP-CoMHE. Table 6.8 shows that all MHE variants consistently improve PN and PN++, while RP-CoMHE and AP-CoMHE again perform the best among all. We demonstrate that CoMHE is generally useful for different types of input data (not limit to images) and different types of neural networks. CoMHE is also useful in graph neural networks (section E.10).

6.7 Concluding Remarks

Since naively minimizing hyperspherical energy yields suboptimal solutions, we propose a novel framework which projects the neurons to suitable spaces and minimizes the energy there. Experiments well validate the superiority of CoMHE.

Part IV:

Hyperspherical Training Paradigm

Inspired by the observation that minimum hyperspherical energy leads to better generalization, we propose a new training paradigm – orthogonal over-parameterized training (OPT) that can provably minimize the hyperspheriacl energy of neurons. We show that OPT is very generic and can improve the empirical generalization of a diverse set of neural networks including convolutional neural networks, point cloud neural networks [158] and graph convolutional networks [161].

The next chapter is based on the following paper:

- *W. Liu*, R. Lin, Z. Liu, J. Rehg, L. Xiong, A. Weller, L. Song. Orthogonal Over-Parameterized Training. *arXiv preprint arXiv:2004.04690*

CHAPTER 7

ORTHOGONAL OVER-PARAMETERIZED TRAINING

7.1 Introduction

The inductive bias encoded in a neural network is generally determined by two major aspects: how the neural network is structured (*i.e.*, network architecture) and how the neural network is optimized (*i.e.*, training algorithm). For the same network architecture, using different training algorithms could lead to a dramatic difference in generalization performance [162, 163] even if the training loss is already close to zero, implying that different training procedures lead to different inductive biases. Therefore, how to effectively train a neural network that can generalize well remains an open challenge.

Recent theories [164, 165, 166, 167, 165] suggest the importance of over-parameterization in linear neural networks. For example, [164] shows that optimizing an underdetermined quadratic objective over a matrix $\boldsymbol{M}$ with gradient descent on a factorization of $\boldsymbol{M}$ leads to an implicit regularization that may improve generalization. There is also strong empirical evidence [168, 33] that over-parameterzing the convolutional filters under some regularity is beneficial to generalization. Our paper aims to leverage the power of over-parameterization and explore more intrinsic structural priors in order to train a well-performing neural network.

Motivated by this goal, we propose a generic orthogonal over-parameterized training (OPT) framework for neural networks. Different from earlier methods, OPT over-parameterizes a neuron $\boldsymbol{w} \in \mathbb{R}^d$ with the multiplication of a learnable layer-shared orthogonal matrix $\boldsymbol{R} \in \mathbb{R}^{d\times d}$ and a fixed randomly initialized weight vector $\boldsymbol{v} \in \mathbb{R}^d$, and it follows that the equivalent weight for the neuron is $\boldsymbol{w} = \boldsymbol{R}\boldsymbol{v}$. Once each element of the neuron weight $\boldsymbol{v}$ has been randomly initialized by a zero-mean Gaussian distribution [38,

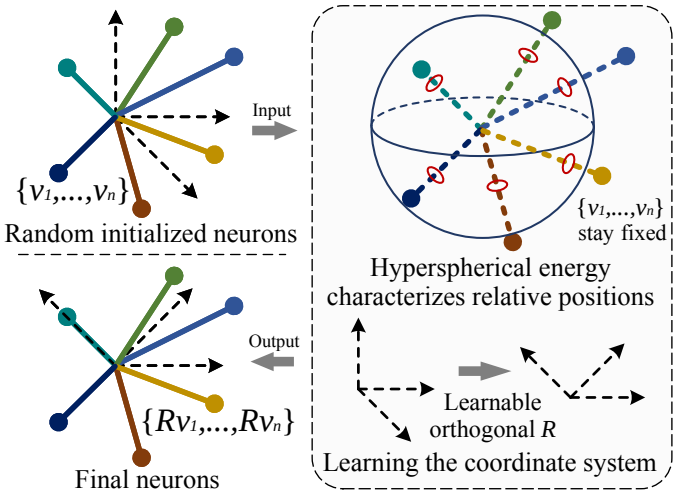


Figure 7.1: Overview of OPT.

82], we fix them throughout the entire training process. Then OPT learns a layer-shared orthogonal transformation $\boldsymbol{R}$ that is applied to all the neurons (in the same layer). An illustration of OPT is given in Figure 7.1. In contrast to standard neural training, OPT decomposes the neuron into two components: an orthogonal transformation $\boldsymbol{R}$ that learns a proper coordinate system and a weight vector $\boldsymbol{v}$ that controls the specific position of the neuron. Essentially, the weights $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n \in \mathbb{R}^d\}$ of different neurons in the same layer determine the relative positions among these neurons, while the layer-shared orthogonal matrix $\boldsymbol{R}$ specifies the coordinate system. This decoupled parameterization enables strong interpretability and flexibility.

Another motivation of OPT comes from an empirical observation [31, 32] that neural networks with lower *hyperspherical energy* generalize better. Hyperspherical energy quantifies the diversity of neurons on a hypersphere, and essentially characterizes the relative positions among neurons via this form of diversity. [31] introduces hyperspherical energy as a regularization in the network but does not guarantee the hyperspherical energy can be effectively minimized (due to the existence of data fitting loss). To address this, we leverage the property of hyperspherical energy that it is independent of the coordinate system in which the neurons live and only depends on their relative positions. Specifically, we prove that, if we randomly initialize the neuron weight $\boldsymbol{v}$ with certain distributions, these

neurons are guaranteed to attain minimum hyperspherical energy in expectation. It follows that OPT maintains the minimum energy during training by learning a coordinate system (*i.e.*, layer-shared orthogonal matrix) for the neurons. Therefore, OPT can well minimize the hyperspherical energy.

We consider several ways to learn the orthogonal transformation. First, we unroll different orthogonalization algorithms such as Gram-Schmidt process, Householder reflection and Löwdin's symmetric orthogonalization. Different unrolled algorithms yield different implicit regularizations to construct the neuron weights. For example, symmetric orthogonalization guarantees that the new orthogonal basis has the least distance in the Hilbert space from the original non-orthogonal basis. Second, we consider to use a special parameterization (*e.g.*Cayley parameterization) to construct the orthogonal matrix, which is more efficient in training. Third, we propose an orthogonal-preserving gradient descent to ensure that the matrix $\boldsymbol{R}$ stays orthogonal after each gradient update. Last, we relax the original optimization problem by making the orthogonality constraint a regularization for the matrix $\boldsymbol{R}$. Different ways of learning the orthogonal transformation may encode different inductive biases.

Moreover, we propose a refinement strategy to further reduce the hyperspherical energy for the randomly initialized neuron weights $\{\boldsymbol{v}, \cdots, \boldsymbol{v}_n\}$. We directly minimize the hyperspherical energy of these random weights as a preprocessing step before training them on actual data. Finally, we provide some theoretical insights and discussions about why OPT may yield better generalization than standard training. We summarize the main advantages of OPT as follows:

- OPT is a generic neural training framework with strong interpretability and flexibility. There are many ways to learn the orthogonal transformation and each one imposes a unique inductive bias.

- OPT is the first training method where the hyperspherical energy is provably minimized, leading to better empirical generalization. OPT reveals that learning a proper

coordinate system is crucial to generalization, and the relative neuron positions are well characterized by hyperspherical energy.

- There is no extra computational cost for OPT-trained neural networks in inference. It has the same inference speed and model size as its standard counterpart. Our experiments also show that OPT performs well on different neural networks [42, 61, 158, 161] and therefore is architecture-agnostic.

7.2 Related Work

Optimization for Deep Learning. A number of first-order optimization algorithms [169, 170, 171, 172, 173, 163] are proposed to improve the empirical convergence and generalization of neural networks. Our work is in parallel with these optimization algorithms, since they can be easily applied to OPT.

Parameterization of Neurons. There are various ways to parameterize a neuron for different applications. [168] over-parameterizes a 2D convolution kernel by combining a 2D kernel of the same size and two additional 1D asymmetric kernels. The resulted convolution kernel has the same effective parameters during testing but more parameters during training. [33] constructs a neuron with a bilinear parameterization and regularizes the bilinear similarity matrix. [174] reparameterizes the neuron matrix with an adaptive fastfood transform to compress model parameters. [175, 176, 177] employ sparse and low-rank structures to construct convolution kernels for a efficient neural network.

Hyperspherical learning. [12, 20, 7] propose to learn representations on hypersphere and show that the angular information in neural networks, in contrast to magnitude information, preserves the most semantic meaning. [31] defines hyperspherical energy that quantifies the diversity of neurons on a hypersphere and empirically shows that the minimum hyperspherical energy improves generalization.

7.3 Orthogonal Over-Parameterized Training

7.3.1 General Framework

OPT parameterizes the neuron as the multiplication of an orthogonal matrix $\boldsymbol{R} \in \mathbb{R}^{d\times d}$ and a neuron weight vector $\boldsymbol{v} \in \mathbb{R}^d$, and the equivalent neuron weight becomes $\boldsymbol{w} = \boldsymbol{R}\boldsymbol{v}$. The output $\hat{y}$ of this neuron can be represented by $\hat{y} = (\boldsymbol{R}\boldsymbol{v})^\top \boldsymbol{x}$ where $\boldsymbol{x} \in \mathbb{R}^d$ is the input vector. In OPT, we fix the randomly initialized neuron weight $\boldsymbol{v}$ and only learn the orthogonal matrix $\boldsymbol{R}$. In contrast, the standard neuron is directly formulated as $\hat{y} = \boldsymbol{v}^\top \boldsymbol{x}$, where the weight vector $\boldsymbol{v}$ is learned in training.

As an illustrative example, we consider a two-layer linear MLP with a loss function $\mathcal{L}$ (*e.g.*, the least squares loss: $\mathcal{L}(e_1, e_2) = (e_1 - e_2)^2$). Specifically, the learning objective of the standard training is $\min_{\{\boldsymbol{v}_i, u_i, \forall i\}} \sum_{j=1}^m \mathcal{L}\big(y, \sum_{i=1}^n u_i \boldsymbol{v}_i^\top \boldsymbol{x}_j\big)$, while differently, our OPT is formulated as

$$\min_{\{\boldsymbol{R}, u_i, \forall i\}} \sum_{j=1}^m \mathcal{L}\big(y, \sum_{i=1}^n u_i (\boldsymbol{R}\boldsymbol{v}_i)^\top \boldsymbol{x}_j\big) \quad \text{s.t. } \boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I} \tag{7.1}$$

where $\boldsymbol{v}_i \in \mathbb{R}^d$ is the i-th neuron in the first layer, and $\boldsymbol{u} = \{u_1, \cdots, u_n\} \in \mathbb{R}^n$ is the output neuron in the second layer. In OPT, each element of $\boldsymbol{v}_i$ is usually sampled from a zero-mean Gaussian distribution, and is fixed throughout the entire training process. In general, OPT learns an orthogonal matrix that is applied to all the neurons instead of learning the individual neuron weight. Note that, we usually do not apply OPT to neurons in the output layer (*e.g.*, $\boldsymbol{u}$ in this MLP example, and the final linear classifiers in CNNs), since it makes little sense to fix a set of random linear classifiers. Therefore, the central problem is how to learn these layer-shared orthogonal matrices.

7.3.2 Hyperspherical Energy Perspective

We delve into OPT from the hyperspherical energy perspective. Following [31], the hyperspherical energy of n neurons is defined as $\boldsymbol{E}(\hat{\boldsymbol{v}}_i|_{i=1}^n) = \sum_{i=1}^n \sum_{j=1,j\neq i}^n \|\hat{\boldsymbol{v}}_i - \hat{\boldsymbol{v}}_j\|^{-1}$ in which $\hat{\boldsymbol{v}}_i = \frac{\boldsymbol{v}_i}{\|\boldsymbol{v}_i\|}$ is the i-th neuron weight projected onto the unit hypersphere $\mathbb{S}^{d-1} = \{\boldsymbol{v} \in \mathbb{R}^d | \, \|\boldsymbol{v}\| = 1\}$. Hyperspherical energy is used to characterize the diversity of n neurons on a unit hypersphere. Assume that we have n neurons in one layer, and we have learned an orthogonal matrix $\boldsymbol{R}$ for these neurons. The hyperspherical energy of these n OPT-trained neurons is given by

$$\boldsymbol{E}(\hat{\boldsymbol{R}}\hat{\boldsymbol{v}}_i|_{i=1}^n) = \sum_{i=1}^n \sum_{j=1,j\neq i}^n \|\boldsymbol{R}\hat{\boldsymbol{v}}_i - \boldsymbol{R}\hat{\boldsymbol{v}}_j\|^{-1} = \sum_{i=1}^n \sum_{j=1,j\neq i}^n \|\hat{\boldsymbol{v}}_i - \hat{\boldsymbol{v}}_j\|^{-1} = \boldsymbol{E}(\hat{\boldsymbol{v}}_i|_{i=1}^n) \quad (7.2)$$

which shows that the hyperspherical energy does not change in OPT. Moreover, [31] proves that minimum hyperspherical energy corresponds to the uniform distribution over the hypersphere. As a result, if the initialization of the neurons in the same layer follows the uniform distribution over the hypersphere, then we can guarantee that the hyperspherical energy is minimal in a probabilistic sense.

Theorem 6. *For the neuron $\boldsymbol{h}=\{h_1, \cdots, h_d\}$ where $h_i, \forall i$ are initialized i.i.d. following a zero-mean Gaussian distribution (i.e., $h_i \sim N(0, \sigma^2)$), the projections onto a unit hypersphere $\hat{\boldsymbol{h}}=\boldsymbol{h}/\|\boldsymbol{h}\|$ where $\|\boldsymbol{h}\|=\sqrt{\Sigma_i h_i^2}$ are uniformly distributed on the unit hypersphere $\mathbb{S}^{d-1}$. The neurons with minimum hyperspherical energy attained asymptotically corresponds to the uniform distribution on $\mathbb{S}^{d-1}$.*

Theorem 6 implies that, if we initialize the neurons in the same layer with zero-mean Gaussians, the corresponding expected hyperspherical energy is guaranteed to be small. It is because the neurons are uniformly distributed on the unit hypersphere and hyperspherical energy quantifies the uniformity on the hypersphere in some sense. More importantly, prevailing neuron initializations such as [82] and [38] are zero-mean Gaussian distribution.

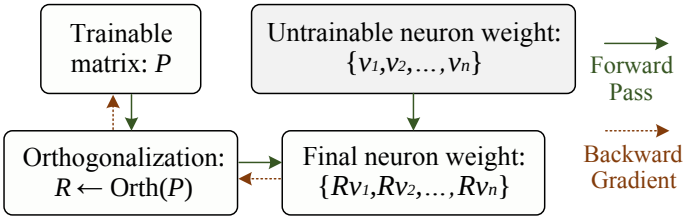


Figure 7.2: Unrolled orthogonalization.

Therefore, our neurons naturally have low hyperspherical energy from the beginning. The appendix in section F.9 gives geometric properties of the random initialized neurons.

7.3.3 Unrolling Orthogonalization Algorithms

To learn the orthogonal transformation, we propose to unroll classic orthogonalization algorithms and embed them into the neural network such that the training is still end-to-end. We need to make every step of the orthogonalization algorithm differentiable, and the training flow is shown in Figure 7.2.

Gram-Schmidt Process. This method takes a linearly independent set and produces an orthogonal set based on it. The Gram-Schmidt Process (GS) usually takes the following steps to orthogonalize a set of vectors $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\} \in \mathbb{R}^{n\times n}$ and obtain an orthogonal set $\{\boldsymbol{e}_1, \cdots, \boldsymbol{e}_i, \cdots, \boldsymbol{e}_n\} \in \mathbb{R}^{n\times n}$. First, when $i = 1$, we have $\boldsymbol{e}_1 = \frac{\tilde{\boldsymbol{e}}_1}{\|\tilde{\boldsymbol{e}}_1\|}$ where $\tilde{\boldsymbol{e}}_1 = \boldsymbol{u}_1$. Then, when $n \geq i \geq 2$, we have $\boldsymbol{e}_i = \frac{\tilde{\boldsymbol{e}}_i}{\|\tilde{\boldsymbol{e}}_i\|}$ where $\tilde{\boldsymbol{e}}_i = \boldsymbol{u}_i - \sum_{j=1}^{i-1} \text{Proj}_{\boldsymbol{e}_j}(\boldsymbol{u}_i)$. $\text{Proj}_{\boldsymbol{b}}(\boldsymbol{a}) = \frac{\langle \boldsymbol{a},\boldsymbol{b}\rangle}{\langle \boldsymbol{b},\boldsymbol{b}\rangle}\boldsymbol{b}$ is the projection operator. We can use modified GS for numerical stability. For better orthogonality, we can unroll an iterative GS [178] with multiple steps.

Householder Reflection. A Householder reflector is defined as $\boldsymbol{H} = \boldsymbol{I} - 2\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}$ where $\boldsymbol{u}$ is perpendicular to the reflection hyperplane. In QR factorization, Householder reflection (HR) is used to transform a (non-singular) square matrix into an orthogonal matrix and an upper triangular matrix. Given a matrix $\boldsymbol{U} = \{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\} \in \mathbb{R}^{n\times n}$, we consider the first column vector $\boldsymbol{u}_1$. We use Householder reflector to transform $\boldsymbol{u}_1$ to $\boldsymbol{e}_1 = \{1, 0, \cdots, 0\}$.

Specifically, we construct $\boldsymbol{H}_1$ as

$$\boldsymbol{H}_1 = \boldsymbol{I} - 2\frac{(\boldsymbol{u}_1 - \|\boldsymbol{u}_1\|\,\boldsymbol{e}_1)(\boldsymbol{u}_1 - \|\boldsymbol{u}_1\|\,\boldsymbol{e}_1)^\top}{\|\boldsymbol{u}_1 - \|\boldsymbol{u}_1\|\,\boldsymbol{e}_1\|^2} \tag{7.3}$$

which is orthogonal. The first column of $\boldsymbol{H}_1\boldsymbol{U}$ becomes $\{\|\boldsymbol{u_1}\|, 0, \cdots, 0\}$. At the k-th step, we can view the sub-matrix $\boldsymbol{U}_{(k:n,k:n)}$ as a new $\boldsymbol{U}$, and use the same procedure to construct the Householder transformation $\tilde{\boldsymbol{H}}_k \in \mathbb{R}^{(n-k)\times(n-k)}$. We construct the final Householder transformation as $\boldsymbol{H}_k = \text{Diag}(\boldsymbol{I}_k, \tilde{\boldsymbol{H}}_k)$. Now we can gradually transform $\boldsymbol{U}$ to an upper triangular matrix with n Householder reflections. Therefore, we have that $\boldsymbol{H}_n \cdots \boldsymbol{H}_2\boldsymbol{H}_1\boldsymbol{U} = \boldsymbol{R}$ where $\boldsymbol{R}$ is the upper triangular matrix (different from the matrix $\boldsymbol{R}$ in Figure 7.2) and the obtained orthogonal set is $\boldsymbol{Q}^\top = \boldsymbol{H}_n \cdots \boldsymbol{H}_2\boldsymbol{H}_1$.

Löwdin's Symmetric Orthogonalization. Let the matrix $\boldsymbol{U} = \{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\} \in \mathbb{R}^{n\times n}$ be a given set of linearly independent vectors in an n-dimensional space. A non-singular linear transformation $\boldsymbol{A}$ can transform the basis $\boldsymbol{U}$ to an orthogonal basis $\boldsymbol{R}$: $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{A}$. The matrix $\boldsymbol{R}$ will be orthogonal if

$$\boldsymbol{R}^\top\boldsymbol{R} = (\boldsymbol{U}\boldsymbol{A})^\top\boldsymbol{U}\boldsymbol{A} = \boldsymbol{A}^\top\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{A} = \boldsymbol{A}^\top\boldsymbol{M}\boldsymbol{A} = \boldsymbol{I} \tag{7.4}$$

where $\boldsymbol{M} = \boldsymbol{U}^\top\boldsymbol{U}$ is the Gram matrix of the given set $\boldsymbol{U}$. We obtain a general solution to the orthogonalization problem via the substitution: $\boldsymbol{A} = \boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{B}$ where $\boldsymbol{B}$ is an arbitrary unitary matrix. The specific choice $\boldsymbol{B} = \boldsymbol{I}$ gives the Löwdin's symmetric orthogonalization (LS): $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{M}^{-\frac{1}{2}}$. We can analytically obtain the symmetric orthogonalization from the singular value decomposition: $\boldsymbol{U} = \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{V}^\top$. Then LS gives $\boldsymbol{R} = \boldsymbol{W}\boldsymbol{V}^\top$ as the orthogonal set for $\boldsymbol{U}$. LS possesses a remarkable property which the other orthogonalizations do not have. The orthogonal set resembles the original set in a nearest-neighbour sense. Specifically, LS guarantees that $\sum_i \|\boldsymbol{R}_i - \boldsymbol{U}_i\|^2$ (where $\boldsymbol{R}_i$ and $\boldsymbol{U}_i$ are the i-th column of $\boldsymbol{R}$ and $\boldsymbol{U}$, respectively) is minimized. Intuitively, LS indicates the gentlest pushing of the directions of the vectors in order to get them orthogonal.

Discussion. These orthogonalization algorithms are fully differentiable and end-to-end trainable. For better orthogonality, these algorithms can be used iteratively and we can unroll them with multiple iterations. Empirically, one-step unrolling already works well. We have also considered Givens rotations to construct the orthogonal matrix, but this requires traversing all lower triangular elements in the original set $\boldsymbol{U}$, which takes $\mathcal{O}(n^2)$ complexity and is very costly. Interestingly, each orthogonalization encodes a unique inductive bias for the resulting neurons by imposing implicit regularizations (*e.g.*, least distance in Hilbert space for LS). More details are given in section F.1.

7.3.4 Orthogonal Parameterization

A convenient way to ensure orthogonality while learning the matrix $\boldsymbol{R}$ is to use a special parameterization that inherently guarantees orthogonality. The exponential parameterization use $\boldsymbol{R} = \exp(\boldsymbol{W})$ (where $\exp(\cdot)$ denotes the matrix exponential) to represent an orthogonal matrix from a skew-symmetric matrix $\boldsymbol{W}$. The Cayley parameterization (CP) is a Padé approximation of the exponential parameterization, and is a more natural choice due to its simplicity. CP uses the following transform to construct an orthogonal matrix $\boldsymbol{R}$ from a skew-symmetric matrix $\boldsymbol{W}$: $\boldsymbol{R} = (\boldsymbol{I} + \boldsymbol{W})(\boldsymbol{I} - \boldsymbol{W})^{-1}$ where $\boldsymbol{W} = -\boldsymbol{W}^\top$. We note that CP only produces the orthogonal matrices with determinant 1, which belong to the special orthogonal group and thus $\boldsymbol{R} \in SO(n)$. Specifically, it suffices to learn the upper or lower triangular of the matrix $\boldsymbol{W}$ with unconstrained optimization to obtain a desired orthogonal matrix $\boldsymbol{R}$. Cayley parameterization does not cover the entire orthogonal group and is less flexible in terms of representation power, which serves as a explicit regularization for the neurons.

7.3.5 Orthogonality-Preserving Gradient Descent

An alternative way to guarantee orthogonality is to modify the gradient update for the matrix $\boldsymbol{R}$. The idea is to initialize $\boldsymbol{R}$ with an arbitrary orthogonal matrix and then ensure

each gradient update is to apply an orthogonal transformation to $\boldsymbol{R}$. It is essentially conducting gradient descent on the Stiefel manifold [179, 180, 181, 182, 183, 184, 185]. Given a matrix $\boldsymbol{U}_{(0)} \in \mathbb{R}^{n\times n}$ that is initialized as an orthogonal matrix, we aim to construct an orthogonal transformation as the gradient update. We use the Cayley transform to compute a parametric curve on the Stiefel manifold $\mathcal{M}_s = \{\boldsymbol{U} \in \mathbb{R}^{n\times n} : \boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}\}$ with a specific metric via a skew-symmetric matrix $\boldsymbol{W}$ and use it as the update rule:

$$\boldsymbol{Y}(\lambda) = (\boldsymbol{I} - \frac{\lambda}{2}\boldsymbol{W})^{-1}(\boldsymbol{I} + \frac{\lambda}{2}\boldsymbol{W})\boldsymbol{U}_{(i)},\ \boldsymbol{U}_{(i+1)} = \boldsymbol{Y}(\lambda) \tag{7.5}$$

where $\hat{\boldsymbol{W}} = \nabla f(\boldsymbol{U}_{(i)})\boldsymbol{U}_{(i)}^\top - \frac{1}{2}\boldsymbol{U}_{(i)}(\boldsymbol{U}_{(i)}^\top \nabla f(\boldsymbol{U}_{(i)}\boldsymbol{U}_{(i)}^\top)$ and $\boldsymbol{W} = \hat{\boldsymbol{W}} - \hat{\boldsymbol{W}}^\top$. $\boldsymbol{U}_{(i)}$ denotes the orthogonal matrix in the i-th iteration. $\nabla f(\boldsymbol{U}_{(i)})$ denotes the original gradient of the loss function *w.r.t.* $\boldsymbol{U}_{(i)}$. We term this gradient update as orthogonal-preserving gradient descent (OGD). To reduce the computational cost of the matrix inverse in Equation 7.5, we use an iterative method [179] to approximate the Cayley transform without matrix inverse. We arrive at the fixed-point iteration from Equation 7.5:

$$\boldsymbol{Y}(\lambda) = \boldsymbol{U}_{(i)} + \frac{\lambda}{2}\boldsymbol{W}\big(\boldsymbol{U}_{(i)} + \boldsymbol{Y}(\lambda)\big) \tag{7.6}$$

which converges to the closed-form Cayley transform with a rate of $o(\lambda^{2+n})$ (n is the iteration number). In practice, we find that two iterations will suffice for a reasonable approximation accuracy.

7.3.6 Relaxation to Orthogonal Regularization

We consider relaxing the original optimization with an orthogonality constraint to an unconstrained optimization with orthogonality regularization (OR). Specifically, we remove the orthogonality constraint, and adopt an orthogonality regularization for $\boldsymbol{R}$, *i.e.*, $\|\boldsymbol{R}^\top \boldsymbol{R} - \boldsymbol{I}\|_F^2$. However, OR cannot guarantee the energy stays unchanged. Taking Equa-

tion 7.1 as an example, we change the objective to

$$\min_{\boldsymbol{R},u_i,\forall i} \sum_{j=1}^{m} \mathcal{L}\big(y, \sum_{i=1}^{n} u_i(\boldsymbol{R}\boldsymbol{v}_i)^\top \boldsymbol{x}_j\big) + \beta\|\boldsymbol{R}^\top\boldsymbol{R} - \boldsymbol{I}\|_F^2 \tag{7.7}$$

where β is a hyperparameter. This serves as an relaxation of the original OPT objective. Note that, OR is imposed to $\boldsymbol{R}$ and is quite different from the existing orthogonal regularization on neurons [12, 29].

7.3.7 Refining the Random Initialization as Preprocessing

Minimizing hyperspherical energy before training. Because we randomly initialize the neurons $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\}$, there exists a variance that makes the hyperspherical energy deviate from the minima even if the hyperspherical energy is minimal in a probabilistic sense. To further reduce the hyperspherical energy, we propose to refine the random initialization by minimizing its hyperspherical energy as a preprocessing step before the OPT training. Specifically, before feeding these neurons to OPT, we first minimize the hyperspherical energy of the initialized neurons with gradient descent (without fitting the training data). Moreover, since the randomly initialized neurons cannot guarantee to get rid of the collinearity redundancy as shown in [31] (*i.e.*, two neurons are on the same line but have opposite directions), we can perform the half-space hyperspherical energy minimization [31].

Normalizing the neurons. The norm of the randomly initialized neurons may have some influence on OPT, serving a role similar to weighting the importance of different neurons. Moreover, the norm makes the hyperspherical energy less expressive to characterize the diversity of neurons, as discussed in subsection 7.4.2. To address this, we propose to normalize the neuron weights such that the weight of each neuron has the unit norm. Because the weights of the neurons $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\}$ are fixed throughout the training process and OPT will not change the norm of the final neurons, we only need to normalize the randomly initialized neuron weights as a preprocessing before the OPT training.

We have comprehensively evaluated both refinements in subsection 7.5.3 and verified their effectiveness. Moreover, we note that OPT still performs well even without these refinements.

7.4 Insights and Discussions

7.4.1 Optimization, Generalization and Inductive Bias

We give some theoretical discussions about why OPT may improve optimization and lead to better generalization. On one hand, [126] proves that once a variant of the hyperspherical energy is small enough (*i.e.*, the neurons are diverse enough) in a one-hidden-layer neural network, the training loss is on the order of the square norm of the gradient and the generalization error will have an additional term $\tilde{\mathcal{O}}(1/\sqrt{m})$ where m is the number of samples. This suggests that SGD-optimized networks with minimum hyperspherical energy (MHE) attained have no spurious local minima. Since OPT guarantees that MHE can be achieved in expectation, OPT-trained networks also enjoy the same theoretical properties. On the other hand, [166, 186, 187, 167, 164] shows that over-parameterization in neural networks can improve the first-order optimization, lead to better generalization, and may impose some implicit regularizations. In the light of this, OPT also introduces over-parameterization in each neuron in the neural network. It can be viewed as a form of local over-parameterization that shares similar spirit with [188]. Specifically, one d-dimensional neuron has d^2+d parameters in OPT, compared to d parameters in the baseline (d^2 parameters are shared across neurons of the same layer). Although OPT uses more parameters to represent a neuron in training, the equivalent number of parameters for a neuron does not change and therefore will affect inference speed once being trained. We further argue that MHE is an effective inductive bias for neural networks that leads to better generalization. As a provable and feasible way to achieve MHE, OPT is particularly useful in practice. Although [31] demonstrates that MHE empirically performs well, why lower hyperspherical energy improves generalization is not fully understood in theory, which remains our future

work.

7.4.2 Discussions

Semi-randomness. OPT fixes the randomly initialized neuron weight vectors and simply learns layer-shared orthogonal matrices, so OPT naturally imposes strong randomness to the neurons. OPT well combines the good generalizability from randomness and the strong approximation power from neural networks. [136, 137, 116] also show that randomness can be beneficial to generalization.

Coordinate system and relative position. OPT shows that learning the coordinate system yields better generalization than learning neuron weights directly. This implies that the coordinate system is crucial to generalization. However, the relative position does not matter only when the hyperspherical energy is sufficiently low, indicating that hyperspherical energy well characterizes the relative positions among neurons. Lower hyperspherical energy generally leads to better generalization.

The effects of neuron norm. Because we will normalize the neuron norm when computing the hyperspherical energy, the effects of neuron norm will not be taken into consideration. Moreover, simply learning the orthogonal matrices will not change the neuron norm. Therefore, the neuron norm will inevitably affect the trained neural network. We use an extreme case as an example to demonstrate the effects. Assume we have N neurons. One of the neurons have norm 1000 and the other neurons have norm 0.01. Then no matter what orthogonal matrices we have learned, the final performance is doomed to be bad. In this case, the hyperspherical energy can still be minimized to a very low value, but it can not capture the norm distribution. Fortunately, such an extreme case is highly unlikely to happen. We are using zero-mean Gaussian distribution to initialize each element of the neuron, and the vector norm will follow a variant of the chi distribution. Therefore every neuron also has the same expected value for the norm, indicating that all neurons have similar norm. In order to completely eliminate the effects of norms, we consider to

Table 7.1: Testing error (%) on CIFAR-100.

Method	FN	LR	CNN-6	CNN-9
Baseline	-	-	37.59	33.55
UPT	N	U	48.47	46.72
UPT	Y	U	42.61	39.38
OPT	N	GS	37.24	32.95
OPT	Y	GS	**33.02**	**31.03**

normalize the neuron weights before the training of the neural network (as a preprocessing step), which has been proposed in subsection 7.3.7.

7.5 Experiments and Results

We aim to show the generalization gain of OPT in multi-layer perceptrons, convolutional neural networks, graph neural networks, and point cloud networks. Experimental details are in section F.3.

7.5.1 Ablation Study and Exploratory Experiment

Necessity of orthogonality. We study whether the orthogonality is necessary for OPT. We use both 6-layer CNN and 9-layer CNN (specified in section F.3) on CIFAR-100. We compare OPT with a baseline with the same network architecture that learns an unconstrained matrix $\boldsymbol{R}$ with only weight decay regularization. We term this baseline as unconstrained over-parameterized training (UPT). "FN" in Table 7.1 denotes whether the randomly initialized neuron weights are fixed throughout the training ("Y" for yes and "N" for no). "LR" denotes whether the learnable transformation $\boldsymbol{R}$ is unconstrained ("U") or orthogonal ("GS" for Gram-Schmidt process). The results in Table 7.1 show that without ensuring orthogonality, UPT performs much worse than OPT that unrolls the Gram-Schmidt process.

Fixed weights vs. learnable weights. From Table 7.1, we can see that using fixed neuron weights is consistently better than learnable neuron weights in both UPT and OPT. It indicates that fixing the neuron weights while learning the transformation matrix $\boldsymbol{R}$ is

Table 7.2: Initial hyperspherical energy.

Mean	Energy	Error (%)
0	**3.5109**	**32.49**
1e-3	3.5117	33.11
1e-2	3.5160	39.51
2e-2	3.5531	53.89
3e-2	3.6761	N/C
5e-2	4.2776	N/C

Table 7.3: Testing error (%) on MNIST.

Method	Normal	Xavier
Baseline	6.05	2.14
OPT (GS)	**5.11**	**1.45**
OPT (HR)	5.31	1.60
OPT (LS)	5.32	1.54
OPT (CP)	5.14	1.49
OPT (OGD)	5.38	1.56
OPT (OR)	5.41	1.78

very beneficial to generalization.

High vs. low hyperspherical energy. We empirically verify that high hyperspherical energy corresponds to inferior generalization performance. To initialize neurons with high hyperspherical energy, we use a random initialization with mean equal to $1e-3$, $1e-2$, $2e-2$, $3e-2$ and $5e-2$. We use CNN-6 to conduct experiments on CIFAR-100. The results in Table 7.2 ("N/C" denotes not converged) show that the network with higher hyperspherical energy is more difficult to converge. Moreover, we find that if the hyperspherical energy is larger than a certain value, then the network cannot converge at all. Note that, when the hyperspherical energy is small (near the minima), a little change in hyperspherical energy (*e.g.*, from 3.5109 to 3.5160) can lead to dramatic generalization gap (*e.g.*, from 32.49% error rate to 39.51%). As expected, one can also observe that higher hyperspherical energy leads to worse generalization.

7.5.2 Multi-Layer Perceptrons

We evaluate all OPT variants on MNIST with a 3-layer MLP. The appendix in section F.3 gives specific settings. Table 7.3 shows the testing error with normal initialization

Table 7.4: Testing error (%) on CIFAR-100.

Method	CNN-6	CNN-9
Baseline	37.59	33.55
HS-MHE	34.97	32.87
OPT (GS)	**33.02**	**31.03**
OPT (HR)	35.67	32.75
OPT (LS)	34.48	31.22
OPT (CP)	33.53	31.28
OPT (OGD)	33.33	31.47
OPT (OR)	34.70	32.63

or Xavier initialization [82]. OPT (GS/HR/LS) are OPT with unrolled orthogonalization algorithms. OPT (CP) denotes OPT with Cayley parameterization. OPT (OGD) is OPT with orthogonal-preserving gradient descent. OPT (OR) denotes OPT with relaxed orthogonal regularization. All OPT variants outperform the baseline by a large margin.

7.5.3 Convolutional Neural Networks

OPT variants. All the OPT variants are evaluated with a plain 6-layer CNN and a plain 9-layer CNN on CIFAR-100. Detailed network architectures are given in section F.3. All the neurons are initialized by [38]. Batch normalization [44] is used by default. Table 7.4 shows that nearly all OPT variants consistently outperform both baseline and the HS-MHE regularization [31] by a significant margin. HS-MHE puts the half-space hyperspherical energy into the loss function and naively minimizes it with stochastic gradients. From the results, we observe that OPT (HR) performs the worse among all OPT variants. In contrast, OPT (GS) achieves the best testing error, implying that Gram-Schmidt process imposes a suitable inductive bias for CNNs on CIFAR-100.

Training without batch normalization. We further evaluate how OPT performs without batch normalization (BN). In specific, we use CNN-6 as our backbone network and test on CIFAR-100. From Table 7.5, one can observe that all OPT variants again outperform both the baseline and HS-MHE [31], validating that OPT is robust and can work reasonably well even without batch normalization. Among all the OPT variants, Cayley parameterization achieves the best testing error with approximately 4% lower than the standard training.

Table 7.5: Testing error (%) on CIFAR-100 without batch normalization.

Method	Error (%)
Baseline	38.95
HS-MHE	36.90
OPT (GS)	35.61
OPT (HR)	37.51
OPT (LS)	35.83
OPT (CP)	**34.88**
OPT (OGD)	35.38
OPT (OR)	N/C

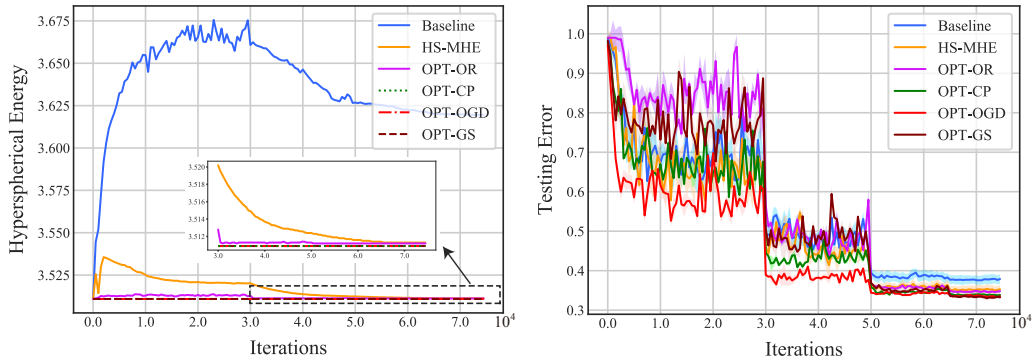


Figure 7.3: Training dynamics on CIFAR-100. Left: Hyperspherical energy vs. iteration. Right: Testing error vs. iteration.

Training dynamics. We also look into how hyperspherical energy and testing error changes while training with OPT. For hyperspherical energy, we can see from Figure 7.3 that the hyperspherical energy of the baseline will increase dramatically at the beginning and then gradually go down, but it still stays in a high value in the end. MHE can effectively reduce the hyperspherical energy at the end of the training. In contrast, all OPT variants maintains minimal hyperspherical energy from the beginning. OPT (GS) and OPT (CP) keep exactly the same hyperspherical energy as the randomly initialized neurons, while OPT (OR) slightly increases the hyperspherical energy due to relaxation. For testing error, all OPT variants converge stably and their final accuracies outperform the others.

Refining neuron initialization. We also evaluate two refinement methods (introduced in subsection 7.3.7) for the neuron initialization. To start with, we consider the hyperspherical energy minimization as a preprocessing for the neuron weights. We conduct the experiment using CNN-6 on CIFAR-100. Specifically, we run a gradient descent for 5k

Table 7.6: Refining hyperspherical energy for OPT.

Method	Original	MHE	HS-MHE
OPT (GS)	33.02	32.99	**32.78**
OPT (LS)	34.48	34.43	**34.37**
OPT (CP)	33.53	33.50	**33.42**
Energy	3.5109	3.5003	**3.4976**

Table 7.7: Testing error (%) of normalized neurons on CIFAR-100.

Method	w/o Norm	w/ Norm
Baseline	37.59	**36.05**
OPT (GS)	33.02	**32.54**
OPT (HR)	35.67	**35.30**
OPT (LS)	34.48	**32.11**
OPT (CP)	33.53	**32.49**
OPT (OGD)	33.37	**32.70**
OPT (OR)	34.70	**33.27**

iterations to minimize the hyperspherical energy of the neuron weights before the training gets started. We also compute the hyperspherical energy (before the training starts and after the preprocessing of energy minimization) in Table 7.6. All the methods start with the same random initialization (*i.e.*, the same random seed), so all the hyperspherical energy starts at 3.5109. After the neuron preprocessing, we have the energy of 3.5003 if we use the MHE as the preprocessing objective and 3.4976 if using the half-space MHE objective. Table 7.6 shows that this refinement can effectively improve the performance of OPT and further reduce the testing error.

Then we experiment the neuron weight normalization in OPT. Normalized neurons make a lot of senses in OPT because the scale of randomly initialized weights does not have any useful property. After randomly initializing the neurons, we directly normalize the scale of the weights to 1. These randomly initialized neurons still possess the important property of achieving minimum hyperspherical energy. Specifically, we use CNN-6 to perform classification on CIFAR-100. The results in Table 7.7 show that normalizing the neurons can greatly improve OPT.

OPT for ResNet. To show that OPT is agnostic to different CNN architectures, we perform classification experiments on CIFAR-100 with both ResNet-20 and ResNet-32 [61].

Table 7.8: Testing error (%) of ResNets on CIFAR-100.

Method	ResNet-20	ResNet-32
Baseline	31.11	30.16
OPT (GS)	30.73	29.56
OPT (CP)	**30.47**	**29.31**

Table 7.9: Testing error (%) on ImageNet.

Method	Top-1 Error	Top-5 Error
Baseline	44.32	21.13
OPT (CP)	**43.67**	**20.26**

We use OPT (GS) and OPT (CP) to train ResNet-20 and ResNet-32. Table 7.8 show that OPT achieves consistent improvements on ResNet compared to the baseline.

ImageNet. We test OPT on the large-scale ImageNet-2012 dataset. Specifically, we use a GPU memory-efficient OPT (CP) to train a plain 10-layer CNN (section F.3) on ImageNet. Our purpose is to validate the superiority of OPT over the corresponding baseline. Table 7.9 shows that OPT (CP) achieves $\sim 0.7\%$ and $\sim 0.9\%$ improvements on top-1 and top-5 error, respectively.

Few-shot learning. To evaluate the cross-task generalization of OPT, we conduct few-shot learning on Mini-ImageNet, following the same experimental setting as [189]. More detailed experimental settings are provided in section F.3. Specifically, we apply OPT with CP to train both the baseline and baseline++ described in [189], and immediately obtain obvious improvements. Therefore, OPT-trained neural nets can generalize well in the challenging few-shot recognition task.

7.5.4 Graph Neural Networks

We also test OPT with graph convolution networks [161] for graph node classification. For fairness, we use the same implementation and hyperparameters as [161]. Training a GCN with OPT is not that straightforward. Specifically, the forward model of GCN is $\boldsymbol{Z} = \text{Softmax}\big(\hat{\boldsymbol{A}} \cdot \text{ReLU}(\hat{\boldsymbol{A}} \cdot \boldsymbol{X} \cdot \boldsymbol{W}_0) \cdot \boldsymbol{W}_1\big)$ where $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{\frac{1}{2}}$. We note that $\boldsymbol{A}$ is the adjacency matrix of the graph, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ ($\boldsymbol{I}$ is an identity matrix), and $\tilde{\boldsymbol{D}} = \sum_j \tilde{\boldsymbol{A}}_{ij}$.

Table 7.10: Few-shot classification on Mini-ImageNet.

Method	5-shot Accuracy (%)
MAML [140]	62.71 ± 0.71
ProtoNet [23]	64.24 ± 0.72
Baseline [189]	62.53 ± 0.69
Baseline w/ OPT	$\mathbf{63.27 \pm 0.68}$
Baseline++ [189]	66.43 ± 0.63
Baseline++ w/ OPT	$\mathbf{66.82 \pm 0.62}$

Table 7.11: Classification accuracy (%) of graph convolutional networks.

Method	Cora	Pubmed
GCN Baseline	81.3	79.0
OPT (CP)	82.0	79.4
OPT (OGD)	**82.3**	**79.5**

$\boldsymbol{X} \in \mathbb{R}^{n \times d}$ is the feature matrix of n nodes in the graph (feature dimension is d). $\boldsymbol{W}_1$ is the weights of the classifiers. $\boldsymbol{W}_1$ is the weights of the classifiers. $\boldsymbol{W}_0$ is the weight matrix of size $d \times h$ where h is the dimension of the hidden space. We treat each column vector of $\boldsymbol{W}_0$ as a neuron, so there are h neurons in total. Then we apply OPT to train these h neurons of dimension d in GCN. We conduct experiments on Cora and Pubmed datsets [190]. We aim to verify the effectiveness of OPT on GCN instead of achieving state-of-the-art performance on this task. Table 7.11 show a reasonable improvement achieved by OPT, validating OPT's universality of training different types of neural networks on different modalities.

7.5.5 Point Cloud Neural Networks

We further test OPT on PointNet [158], a type of neural network that takes raw point clouds as input and classify them based on semantics. To simplify the comparison and remove all the bells and whistles, we use a vanilla PointNet (without T-Net) as our backbone network. We apply OPT to train the MLPs in PointNet. We follow the same experimental settings as [158] and evaluate on the ModelNet-40 dataset [160]. From Table 7.12, we can observe that OPT achieves better accuracy than the PointNet baseline, and OPT (CP) achieves nearly 0.8% improvement. It is in fact significant because we do not add any

Table 7.12: Point Cloud classification on ModelNet-40.

Method	Accuracy (%)
PointNet Baseline	87.1
OPT (GS)	**87.23**
OPT (CP)	**87.86**

additional parameters to the network. The improvement on PointNet further validates that OPT is a generic and effective training framework.

7.6 Concluding Remarks

This paper proposes a novel and general training framework for neural networks. OPT over-parameterizes neurons with neuron weights (randomly initialized and fixed) and a layer-shared orthogonal matrix (learnable). OPT provably achieves minimum hyperspherical energy and maintains the energy during training. We give theoretical insights and extensive empirical evidences to validate OPT's superiority.

CHAPTER 8
CONCLUSION

In this dissertation, we present a unified deep representation learning framework on hypersphere, which inherently introduces a novel hyperspherical inductive bias into deep neural networks. We show that our framework is well motivated from both empirical observations and theories. We discuss our framework from four distinct perspectives: (1) learning objectives on hypersphere; (2) neural architectures on hypersphere; (3) regularizations on hypersphere; (4) hyperspherical training paradigm. From the first three perspectives, we explain how we can utilize the idea of hyperspherical learning to revisit and reinvent corresponding components in deep learning. From the last perspective, we propose a general neural training framework that is heavily inspired by hyperspherical learning. We conduct comprehensive experiments on many applications to demonstrate that our deep hyperspherical learning framework yields better generalization, faster convergence and stronger adversarial robustness compared to the standard deep learning framework.

The deep hyperspherical learning framework essentially aims to address the following two fundamental questions:

- Is hyperspherical inductive bias useful in deep learning?
- How can we effectively incorporate hyperspherical inductive bias into deep learning?

For the first question, we provide quite a few applications in the dissertation, such as face recognition, where hyperspherical inductive bias can achieve state-of-the-art performance. Besides that, there are a number of fundamental advantages to incorporate hyperspherical inductive bias into deep learning, such as better generalization, faster convergence and stronger robustness. We believe that hyperspherical inductive bias is generally useful

in deep learning, which has also been verified by a lot of recent studies [12, 10, 15, 13, 14, 24, 22, 25].

For the second question, our deep hyperspherical learning framework just provides a feasible way to incorporate hyperspherical inductive bias into deep learning. There are a number of alternative ways to achieve this goal. From this perspective, our framework can serve as a good example to inspire more future studies to combine hyperspherical inductive bias to deep learning.

To conclude, our deep hyperspherical learning framework makes the very first effort to perform deep learning with hyperspherical inductive bias. The dissertation shows many empirical evidences to support that hyperspherical inductive bias is very effective to improve the generalization, convergence and robustness of deep neural networks.

Appendices

APPENDIX A

ADDITIONAL RESULTS IN CHAPTER 2

A.1 The intuition of removing the last ReLU

Standard CNNs usually connect ReLU to the bottom of FC1, so the learned features will only distribute in the non-negative range $[0, +\infty)$, which limits the feasible learning space (angle) for the CNNs. To address this shortcoming, both SphereFace and [9] first propose to remove the ReLU nonlinearity that is connected to the bottom of FC1 in SphereFace networks. Intuitively, removing the ReLU can greatly benefit the feature learning, since it provides larger feasible learning space (from angular perspective).

Visualization on MNIST. Figure A.1 shows the 2-D visualization of feature distributions in MNIST with and without the last ReLU. One can observe with ReLU the 2-D feature could only distribute in the first quadrant. Without the last ReLU, the learned feature distribution is much more reasonable.

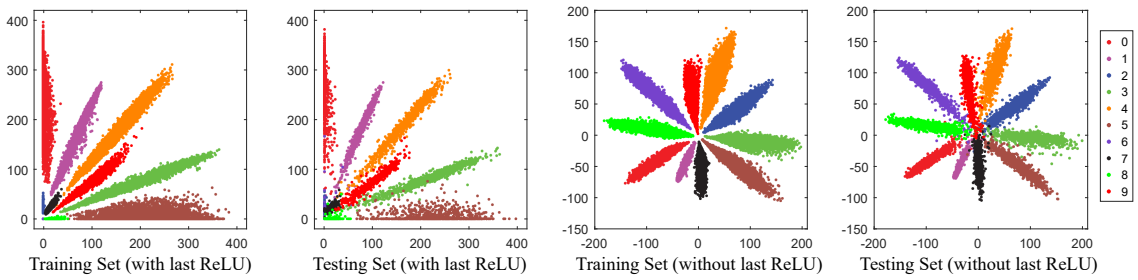


Figure A.1: 2-D visualization before and after removing the last ReLU.

A.2 Normalizing the weights could reduce the prior caused by the training data imbalance

We have emphasized in section 2.3 that normalizing the weights can give better geometric interpretation. Besides this, we also justify why we want to normalize the weights

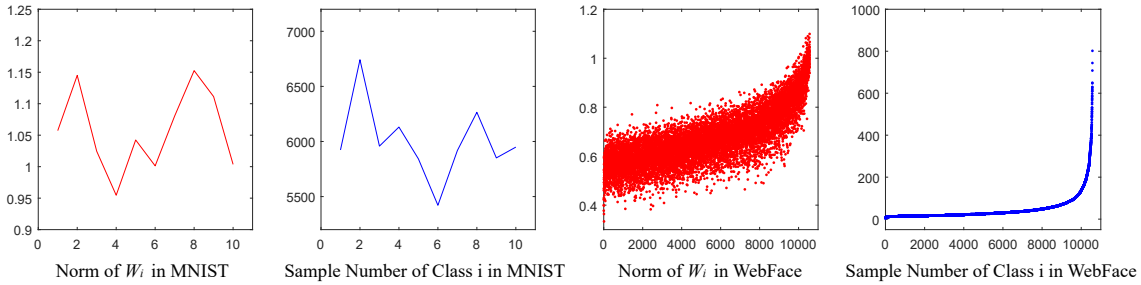


Figure A.2: Norm of $\boldsymbol{W}_i$ and sample number of class i in MNIST dataset and CASIA-WebFace dataset.

from a different perspective. We find that normalizing the weights can implicitly reduce the prior brought by the training data imbalance issue (e.g., the long-tail distribution of the training data). In other words, we argue that normalizing the weights can partially address the training data imbalance problem.

We have an empirical study on the relation between the sample number of each class and the 2-norm of the weights corresponding to the same class (the i-th column of $\boldsymbol{W}$ is associated to the i-th class). By computing the norm of $\boldsymbol{W}_i$ and sample number of class i with respect to each class (see Figure A.2), we find that the larger sample number a class has, the larger the associated norm of weights tends to be. We argue that the norm of weights $\boldsymbol{W}_i$ with respect to class i is largely determined by its sample distribution and sample number. Therefore, norm of weights $\boldsymbol{W}_i, \forall i$ can be viewed as a learned prior hidden in training datasets. Eliminating such prior is often beneficial to face verification. This is because face verification requires to test on a dataset whose idenities can not appear in training datasets, so the prior from training dataset should not be transferred to the testing. This prior may even be harmful to face verification performance. To eliminate such prior, we normalize the norm of weights of FC2[1].

[1]FC2 refers to the fully connected layer in the softmax loss (or A-Softmax loss).

A.3 Empirical experiment of zeroing out the biases

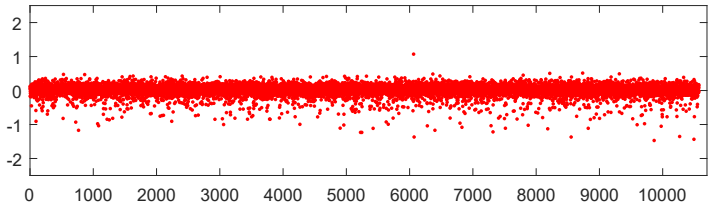


Figure A.3: Biases of last fully connected layer learned in CASIA-WebFace dataset.

Standard CNNs usually preserve the bias term in the fully connected layers, but these bias terms make it difficult to analyze the proposed A-Softmax loss. This is because SphereFace aims to optimize the angle and produce the angular margin. With bias of FC2, the angular geometry interpretation becomes much more difficult to analyze. To facilitate the analysis, we zero out the bias of FC2 following [9]. By setting the bias of FC2 to zero, the A-Softmax loss has clear geometry interpretation and therefore becomes much easier to analyze. We show all the biases of FC2 from a CASIA-pretrained model in Figure A.3. One can observe that the most of the biases are near zero, indicating these biases are not necessarily useful for face verification.

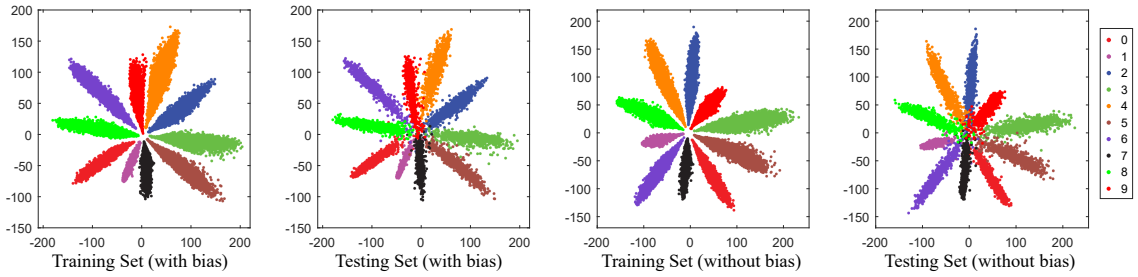


Figure A.4: 2-D visualization with and without bias of last fully connected layer in MNIST.

Visualization on MNIST. We visualize the 2-D feature distribution in MNIST dataset with and without bias in Figure A.4. One can observe that zeroing out the bias has no direct influence on the feature distribution. The features learned with and without bias can both make full use of the learning space.

A.4 2D visualization of A-Softmax loss on MNIST

We visualize the 2-D feature distribution on MNIST in Figure A.5. It is obvious that with larger m the learned features become much more discriminative due to the larger inter-class angular margin. Most importantly, the learned discriminative features also generalize really well in the testing set.

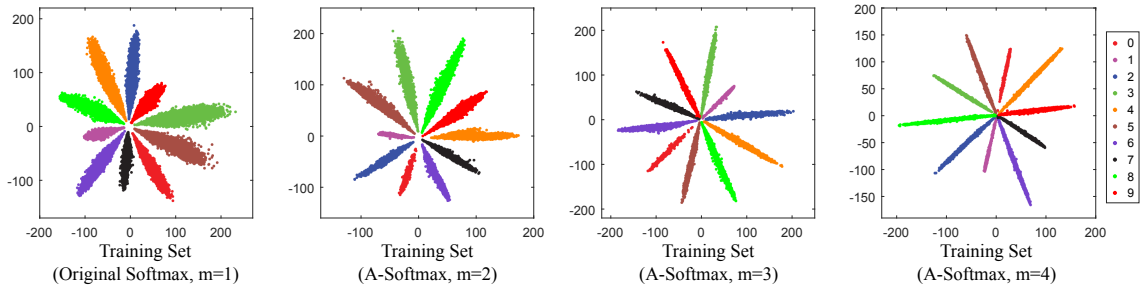


Figure A.5: 2-D MNIST visualization of features learned by the softmax loss and the A-Softmax loss ($m = 2, 3, 4$).

A.5 Angular Fisher score for evaluating the feature discriminativeness and ablation study on our proposed modifications

We first propose an angular Fisher score for evaluating the feature discriminativeness in angular margin feature learning. The angular Fisher score (AFS) is defined by

$$AFS = \frac{S_w}{S_b} \tag{A.1}$$

where the within-class scatter value is defined as $S_w = \sum_i \sum_{x_j \in X_i} (1 - \cos\langle x_j, m_i \rangle)$ and the between-class scatter value is defined as $S_b = \sum_i n_i (1 - \cos\langle m_i, m \rangle)$. X_i is the i-th class samples, m_i is the mean vector of features from class i, m is the mean vector of the whole dataset, and n_i is the sample number of class i. In general, the lower the fisher value is, the more discriminative the features are.

Next, we perform a comprehensive ablation study on all the proposed modifications: removing last ReLU, removing Biases, normalizing weights and applying A-Softmax loss.

The experiments are performed using the 4-layer CNN described in Table 2.7. The models are trained on CASIA dataset and tested on LFW dataset. The setting is exactly the same as the LFW experiment in section 2.3. As shown in Table Table A.1, we could observe that all our modification leads to peformance improvement and our A-Softmax could greatly increase the angular feature discriminativeness.

Table A.1: Verification accuracy (%) on LFW dataset.

CNN	Remove Last ReLU	Remove Biases	Normalize Weights	A-Softmax	Accuracy	Angular Fisher Score
A	No	No	No	No	95.13	0.3477
B	Yes	No	No	No	96.37	0.2835
C	Yes	Yes	No	No	96.40	0.2815
D	Yes	Yes	Yes	No	96.63	0.2462
E	Yes	Yes	Yes	Yes (m=2)	97.67	0.2277
F	Yes	Yes	Yes	Yes (m=3)	97.82	0.1791
G	Yes	Yes	Yes	Yes (m=4)	**98.20**	**0.1709**

A.6 Experiments on MegaFace with different convolutional layers

We also perform the experiment on MegaFace dataset with CNN of different convolutional layers. The results in Table A.2 show that the A-Softmax loss could make best use of the network capacity. With more convolutional layers, the A-Softmax loss (i.e., SphereFace) performs better. Most notably, SphereFace with only 4 convolutional layer could peform better than the softmax loss with 64 convolutional layers, which validates the superiority of our A-Softmax loss.

Table A.2: Performance (%) on MegaFace challenge with different convolutional layers. TAR and FAR denote True Accept Rate and False Accept Rate respectively. For all the SphereFace models, we use $m = 4$. With larger m and proper network optimization, the performance could potentially keep increasing.

Method	protocol	Rank-1 Id. Acc. with 1M distractors	Ver. TAR for 10^{-6} FAR
Softmax Loss (64 conv layers)	Small	54.855	65.925
SphereFace (4 conv layers)	Small	57.529	68.547
SphereFace (10 conv layers)	Small	65.335	78.069
SphereFace (20 conv layers)	Small	69.623	83.159
SphereFace (36 conv layers)	Small	71.257	84.052
SphereFace (64 conv layers)	Small	**72.729**	**85.561**

A.7 The annealing optimization strategy for A-Softmax loss

The optimization of the A-Softmax loss is similar to the L-Softmax loss [9]. We use an annealing optimization strategy to train the network with A-Softmax loss. To be simple, the annealing strategy is essentially supervising the newtork from an easy task (*i.e.*, large λ) gradually to a difficult task (*i.e.*, small λ). Specifically, we let $f_{y_i} = \frac{\lambda\|\boldsymbol{x}_i\|\cos(\theta_{y_i})+\|\boldsymbol{x}_i\|\psi(\theta_{y_i})}{1+\lambda}$ and start the stochastic gradient descent initially with a very large λ (it is equivalent to optimizing the original softmax). Then we gradually reduce λ during training. Ideally λ can be gradually reduced to zero, but in practice, a small value will usually suffice. In most of our face experiments, decaying λ to 5 has already lead to impressive results. Smaller λ could potentially yield a better performance but is also more difficult to train.

A.8 Details of the 3-patch ensemble strategy in MegaFace challenge

We adopt a common strategy to perform the 3-patch ensemble, as shown in Figure A.6. Although using more patches could keep increasing the performance, but considering the tradeoff between efficiency and accuracy, we use 3-patch simple concatenation ensemble (without the use of PCA). The 3 patches can be selected by cross-validation. The 3 patches we use in section 2.3 are exactly the same as in Figure A.6.

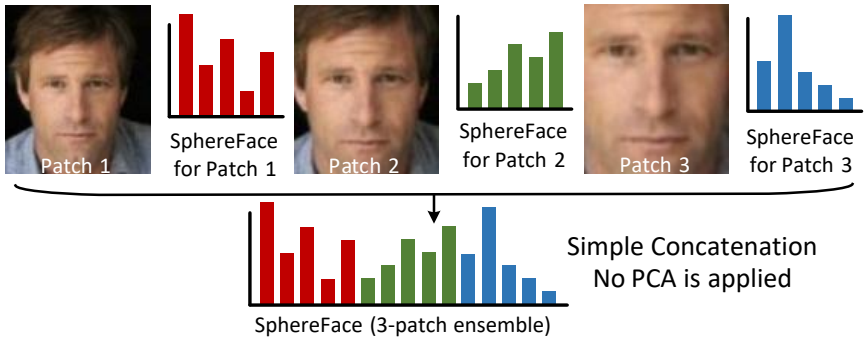


Figure A.6: 3-Patch ensembles in SphereFace for MegaFace challenge.

APPENDIX B

ADDITIONAL RESULTS AND PROOFS IN CHAPTER 3

B.1 Network Architectures

Table B.1: Our plain CNN architectures with different convolutional layers. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers. E.g., [3×3, 64]×3 denotes 3 cascaded convolution layers with 64 filters of size 3×3.

Layer	CNN-3	CNN-9	CNN-18	CNN-45	CNN-60	CNN-69
Conv1.x	[3×3, 64]×1	[3×3, 64]×3	[3×3, 64]×6	[3×3, 64]×15	[3×3, 64]×20	[3×3, 64]×23
Pool1	2×2 Max Pooling, Stride 2					
Conv2.x	[3×3, 96]×1	[3×3, 96]×3	[3×3, 96]×6	[3×3, 96]×15	[3×3, 96]×20	[3×3, 96]×23
Pool2	2×2 Max Pooling, Stride 2					
Conv3.x	[3×3, 128]×1	[3×3, 128]×3	[3×3, 128]×6	[3×3, 128]×15	[3×3, 128]×20	[3×3, 128]×23
Pool3	2×2 Max Pooling, Stride 2					
Fully Connected	256	256	256	256	256	256

Table B.2: Our ResNet architectures with different convolutional layers. Conv0.x, Conv1.x, Conv2.x, Conv3.x and Conv4.x denote convolution units that may contain multiple convolutional layers, and residual units are shown in double-column brackets. Conv1.x, Conv2.x and Conv3.x usually operate on different size feature maps. These networks are essentially the same as [2], but some may have different number of filters in each layer. The downsampling is performed by convolutions with a stride of 2. E.g., [3×3, 64]×4 denotes 4 cascaded convolution layers with 64 filters of size 3×3, and S2 denotes stride 2.

Layer	ResNet-32 for Section 4.2	ResNet-32 for Section 4.3	ResNet-18 for Section 4.6
Conv0.x	N/A	N/A	[7×7, 256], Stride 2 3×3, Max Pooling, Stride 2
Conv1.x	[3×3, 64]×1 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 5$	[3×3, 96]×1 $\begin{bmatrix} 3\times3, 96 \\ 3\times3, 96 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$
Conv2.x	$\begin{bmatrix} 3\times3, 96 \\ 3\times3, 96 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 192 \\ 3\times3, 192 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$
Conv3.x	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 384 \\ 3\times3, 384 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 768 \\ 3\times3, 768 \end{bmatrix} \times 2$
Conv4.x	N/A	N/A	$\begin{bmatrix} 3\times3, 1024 \\ 3\times3, 1024 \end{bmatrix} \times 2$
	Average Pooling		

B.2 Experimental Details for Imagenet-2012

For the input data of the Imagenet-2012 experiment, we only use the minimum data augmentation. Specifically, we first resize the images to 256×256 resolution and then randomly crop patches of size 224×224 from the resized images. Besides that, we also randomly flip the image horizontally. For SphereResNet-18-v1, we use the cosine SphereConv and the cosine W-Softmax loss. For SphereResNet-18-v2, we use the cosine SphereConv and the softmax loss. Generally, we find that the standard softmax loss and all kinds of W-Softmax loss usually have similar empirical performance. Note that, we could obtain better performance by using the other SphereConvs (sigmoid SphereConv with $k = 0.3$ is a good choice), but it requires more GPU memory. Due to the width of our architecture and the limitation of GPU memory, the mini-batch size is set to 40 for all methods in the Imagenet-2012 experiment.

B.3 More Discussions for Sphere-normalized Softmax Loss

The sphere-normalized softmax (S-Softmax) loss is essentially applying the SphereConv to the fully connected layer in the softmax loss. However, simply applying the SphereConv can not make such loss work, because this loss function is difficult to converge in practice. To address this, we rescale the logit output of the S-Softmax loss with a scaling factor s. Therefore, the output range is changed from $[-1, 1]$ to $[-s, s]$. Typically, setting s from 10 to 70 works pretty well in practice. We could also use the cross-validation strategy to set the hyperparameter s.

B.4 Proofs of Lemmas

B.4.1 Proof of Lemma 1

The gradient is

$$\nabla \mathcal{G}(\boldsymbol{U}, \boldsymbol{V}) = \begin{bmatrix} \nabla_{\boldsymbol{U}} \mathcal{G}(\boldsymbol{U}, \boldsymbol{V}) \\ \nabla_{\boldsymbol{V}} \mathcal{G}(\boldsymbol{U}, \boldsymbol{V}) \end{bmatrix} = \begin{bmatrix} (\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F})\boldsymbol{V} \\ (\boldsymbol{V}\boldsymbol{U}^\top - \boldsymbol{F}^\top)\boldsymbol{U} \end{bmatrix}$$

The Hessian matrix is

$$\begin{aligned} \nabla^2 \mathcal{G}(\boldsymbol{U}, \boldsymbol{V}) &= \begin{bmatrix} \nabla^2_{\boldsymbol{U}} \mathcal{G}(\boldsymbol{U}, \boldsymbol{V}) & \nabla^2_{\boldsymbol{U},\boldsymbol{V}} \mathcal{G}(\boldsymbol{U}, \boldsymbol{V}) \\ \nabla^2_{\boldsymbol{V},\boldsymbol{U}} \mathcal{G}(\boldsymbol{U}, \boldsymbol{V}) & \nabla^2_{\boldsymbol{V}} \mathcal{G}(\boldsymbol{U}, \boldsymbol{V}) \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{V}^\top \boldsymbol{V} \otimes \boldsymbol{I}_n & (\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F}) \otimes \boldsymbol{I}_k + \boldsymbol{U} \boxtimes \boldsymbol{V} \\ (\boldsymbol{V}\boldsymbol{U}^\top - \boldsymbol{F}^\top) \otimes \boldsymbol{I}_k + \boldsymbol{V} \boxtimes \boldsymbol{U} & \boldsymbol{U}^\top \boldsymbol{U} \otimes \boldsymbol{I}_m \end{bmatrix}, \end{aligned} \tag{B.1}$$

where $\boldsymbol{I}_n$ is an $n \times n$ identity matrix for any integer n, given matrices $\boldsymbol{A} \in \mathbb{R}^{n \times r}$ and $\boldsymbol{B} \in \mathbb{R}^{m \times k}$ with $\boldsymbol{A}_{:,i}$ denoting the i-th column of $\boldsymbol{A}$, $\boldsymbol{A} \boxtimes \boldsymbol{B} \in \mathbb{R}^{nk \times mr}$ is defined as

$$\boldsymbol{A} \boxtimes \boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}_{:,1}\boldsymbol{B}_{:,1}^\top & \boldsymbol{A}_{:,2}\boldsymbol{B}_{:,1}^\top & \cdots & \boldsymbol{A}_{:,r}\boldsymbol{B}_{:,1}^\top \\ \boldsymbol{A}_{:,1}\boldsymbol{B}_{:,2}^\top & \boldsymbol{A}_{:,2}\boldsymbol{B}_{:,2}^\top & \cdots & \boldsymbol{A}_{:,r}\boldsymbol{B}_{:,2}^\top \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}_{:,1}\boldsymbol{B}_{:,k}^\top & \boldsymbol{A}_{:,2}\boldsymbol{B}_{:,k}^\top & \cdots & \boldsymbol{A}_{:,r}\boldsymbol{B}_{:,k}^\top \end{bmatrix}.$$

At a global optimum, we have $\boldsymbol{U}\boldsymbol{V}^\top = \boldsymbol{F}$. Then it is easy to see that for any real c, if $\tilde{\boldsymbol{U}} = c\boldsymbol{U}$ and $\tilde{\boldsymbol{V}} = \boldsymbol{V}/c$, then we have

$$\nabla^2 \mathcal{G}(\tilde{\boldsymbol{U}}, \tilde{\boldsymbol{V}}) = \begin{bmatrix} \frac{1}{c^2}\boldsymbol{V}^\top \boldsymbol{V} \otimes \boldsymbol{I}_n & \boldsymbol{U} \boxtimes \boldsymbol{V} \\ \boldsymbol{V} \boxtimes \boldsymbol{U} & c^2 \boldsymbol{U}^\top \boldsymbol{U} \otimes \boldsymbol{I}_m \end{bmatrix}$$

We have that at a global optimal point, $\nabla^2\mathcal{G}(\boldsymbol{U},\boldsymbol{V})$ is a positive semidefinite matrix with the smallest eigenvalue equal to 0. Specifically, due to the existence of the invariance, i.e., $\boldsymbol{U}\boldsymbol{V}^\top = \boldsymbol{U}\boldsymbol{R}(\boldsymbol{V}\boldsymbol{R})^\top$ for any orthogonal matrix $\boldsymbol{R} \in \mathbb{R}^{r\times r}$, there are $r(r-1)/2$ number of eigenvectors of $\nabla^2\mathcal{G}(\boldsymbol{U},\boldsymbol{V})$ at $\boldsymbol{U}\boldsymbol{V}^\top = \boldsymbol{F}$ corresponding to 0 eigenvalue [83]. Then for any $c > 1$, we have

$$\begin{aligned}\mathrm{Tr}(\nabla^2\mathcal{G}(\tilde{\boldsymbol{U}},\tilde{\boldsymbol{V}})) &= \frac{1}{c^2}\mathrm{Tr}(\boldsymbol{V}^\top\boldsymbol{V}\otimes\boldsymbol{I}_n) + c^2\mathrm{Tr}(\boldsymbol{U}^\top\boldsymbol{U}\otimes\boldsymbol{I}_m)\\ &\geq \frac{c^2}{2}\left(\mathrm{Tr}(\boldsymbol{V}^\top\boldsymbol{V}\otimes\boldsymbol{I}_n) + \mathrm{Tr}(\boldsymbol{U}^\top\boldsymbol{U}\otimes\boldsymbol{I}_m)\right) = \frac{c^2}{2}\mathrm{Tr}(\nabla^2\mathcal{G}(\boldsymbol{U},\boldsymbol{V})).\end{aligned}$$

This indicates that the largest eigenvalue of $\nabla^2\mathcal{G}(\tilde{\boldsymbol{U}},\tilde{\boldsymbol{V}})$ is on the order of $\Theta(c^2)$ times the largest eigenvalue of $\nabla^2\mathcal{G}(\boldsymbol{U},\boldsymbol{V})$ following the perturbation bound analysis [191] and $\boldsymbol{U}$ and $\boldsymbol{V}$ are balanced. Using a similar idea, the smallest nonzero eigenvalue of $\nabla^2\mathcal{G}(\tilde{\boldsymbol{U}},\tilde{\boldsymbol{V}})$ is no greater than the smallest nonzero eigenvalue of $\nabla^2\mathcal{G}(\boldsymbol{U},\boldsymbol{V})$, which results in our claim on the restricted condition number.

B.4.2 Proof of Lemma 2

The gradient of $\mathcal{G}_S(\boldsymbol{U},\boldsymbol{V})$ is

$$\nabla\mathcal{G}_S(\boldsymbol{U},\boldsymbol{V}) = \begin{bmatrix}\nabla_{\boldsymbol{U}}\mathcal{G}_S(\boldsymbol{U},\boldsymbol{V})\\ \nabla_{\boldsymbol{V}}\mathcal{G}_S(\boldsymbol{U},\boldsymbol{V})\end{bmatrix} \text{ with}$$

$$\nabla_{\boldsymbol{U}}\mathcal{G}_S(\boldsymbol{U},\boldsymbol{V}) = \boldsymbol{D}_{\boldsymbol{U}}(\boldsymbol{D}_{\boldsymbol{U}}\boldsymbol{U}\boldsymbol{V}^\top\boldsymbol{D}_{\boldsymbol{V}} - \boldsymbol{F})\boldsymbol{D}_{\boldsymbol{V}}\boldsymbol{V} - \left(\boldsymbol{D}^3_{\boldsymbol{U}}(\boldsymbol{D}_{\boldsymbol{U}}\boldsymbol{U}\boldsymbol{V}^\top\boldsymbol{D}_{\boldsymbol{V}} - \boldsymbol{F})\circledast_k(\boldsymbol{U}\boldsymbol{V}^\top\boldsymbol{D}_{\boldsymbol{V}})\right)\odot\boldsymbol{U},$$

$$\nabla_{\boldsymbol{V}}\mathcal{G}_S(\boldsymbol{U},\boldsymbol{V}) = \boldsymbol{D}_{\boldsymbol{V}}(\boldsymbol{D}_{\boldsymbol{V}}\boldsymbol{V}\boldsymbol{U}^\top\boldsymbol{D}_{\boldsymbol{U}} - \boldsymbol{F}^\top)\boldsymbol{D}_{\boldsymbol{U}}\boldsymbol{U} - \left(\boldsymbol{D}^3_{\boldsymbol{V}}(\boldsymbol{D}_{\boldsymbol{V}}\boldsymbol{V}\boldsymbol{U}^\top\boldsymbol{D}_{\boldsymbol{U}} - \boldsymbol{F}^\top)\circledast_k(\boldsymbol{V}\boldsymbol{U}^\top\boldsymbol{D}_{\boldsymbol{U}})\right)\odot\boldsymbol{V},$$

Note that after each iteration of SGD, we perform the entry-wise normalization for both

U and $\boldsymbol{V}$, which means $\boldsymbol{D_U} = I_n$ and $\boldsymbol{D_V} = \boldsymbol{I}_m$. Then the gradient of $\mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V})$ is

$$\begin{aligned}\nabla \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) &= \begin{bmatrix} \nabla_{\boldsymbol{U}} \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) \\ \nabla_{\boldsymbol{V}} \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) \end{bmatrix} \\ &= \begin{bmatrix} (\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F})\boldsymbol{V} - \left((\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F}) \circledast_k (\boldsymbol{U}\boldsymbol{V}^\top)\right) \odot \boldsymbol{U} \\ (\boldsymbol{V}\boldsymbol{U}^\top - \boldsymbol{F}^\top)\boldsymbol{U} - \left((\boldsymbol{V}\boldsymbol{U}^\top - \boldsymbol{F}^\top) \circledast_k (\boldsymbol{V}\boldsymbol{U}^\top)\right) \odot \boldsymbol{V} \end{bmatrix},\end{aligned}$$

where given matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n \times m}$ with $\boldsymbol{A}_{:,i}$ denoting the i-th column of $\boldsymbol{A}$, $\boldsymbol{A} \odot \boldsymbol{B} \in \mathbb{R}^{n \times m}$ is the Hadamard (pointwise) product, and the operation $\boldsymbol{A} \circledast_k \boldsymbol{B} \in \mathbb{R}^{n \times k}$ is defined as

$$\boldsymbol{A} \circledast_k \boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}_{1,:}\boldsymbol{B}_{1,:}^\top \\ \boldsymbol{A}_{2,:}\boldsymbol{B}_{2,:}^\top \\ \vdots \\ \boldsymbol{A}_{n,:}\boldsymbol{B}_{n,:}^\top \end{bmatrix} \mathbf{1}_{1 \times k},$$

where $\mathbf{1}_{1 \times k}$ is a $1 \times k$ vector with all entries equal to 1.

Consequently, the Hessian matrix is

$$\nabla^2 \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) = \begin{bmatrix} \nabla^2_{\boldsymbol{U}} \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) & \nabla^2_{\boldsymbol{U},\boldsymbol{V}} \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) \\ \nabla^2_{\boldsymbol{V},\boldsymbol{U}} \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) & \nabla^2_{\boldsymbol{V}} \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) \end{bmatrix} \text{ with}$$

$$\begin{aligned}\nabla^2_{\boldsymbol{U}} \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) &= \boldsymbol{V}^\top \boldsymbol{V} \otimes \boldsymbol{I}_n - \operatorname{diag}\left(\operatorname{vec}\left((\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F}) \circledast_k (\boldsymbol{U}\boldsymbol{V}^\top)\right)\right) \\ &\quad - \begin{bmatrix} \operatorname{diag}\left(\boldsymbol{U}_{:,1} \odot ((2\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F})\boldsymbol{V}_{:,1})\right) & \cdots & \operatorname{diag}\left(\boldsymbol{U}_{:,1} \odot ((2\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F})\boldsymbol{V}_{:,k})\right) \\ \vdots & \ddots & \vdots \\ \operatorname{diag}\left(\boldsymbol{U}_{:,k} \odot ((2\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F})\boldsymbol{V}_{:,1})\right) & \cdots & \operatorname{diag}\left(\boldsymbol{U}_{:,k} \odot ((2\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F})\boldsymbol{V}_{:,k})\right) \end{bmatrix}\end{aligned}$$

$$\begin{aligned}\nabla^2_{\boldsymbol{U},\boldsymbol{V}} \mathcal{G}_S(\boldsymbol{U}, \boldsymbol{V}) &= \boldsymbol{I}_k \otimes (\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F}) + \boldsymbol{U} \boxtimes \boldsymbol{V} \\ &\quad - \begin{bmatrix} (2\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F}) \odot ((\boldsymbol{U}_{:,1} \odot \boldsymbol{U}_{:1})1_{1 \times m}) & \cdots & (2\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F}) \odot ((\boldsymbol{U}_{:,1} \odot \boldsymbol{U}_{:k})\mathbf{1}_{1 \times m}) \\ \vdots & \ddots & \vdots \\ (2\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F}) \odot ((\boldsymbol{U}_{:,k} \odot \boldsymbol{U}_{:1})\mathbf{1}_{1 \times m}) & \cdots & (2\boldsymbol{U}\boldsymbol{V}^\top - \boldsymbol{F}) \odot ((\boldsymbol{U}_{:,k} \odot \boldsymbol{U}_{:k})\mathbf{1}_{1 \times m}) \end{bmatrix}\end{aligned}$$

$$\nabla^2_{\boldsymbol{V},\boldsymbol{U}}\mathcal{G}_S(\boldsymbol{U},\boldsymbol{V}) = \left(\nabla^2_{\boldsymbol{U},\boldsymbol{V}}\mathcal{G}_S(\boldsymbol{U},\boldsymbol{V})\right)^\top$$

$$\nabla^2_{\boldsymbol{V}}\mathcal{G}_S(\boldsymbol{U},\boldsymbol{V}) = \boldsymbol{U}^\top\boldsymbol{U}\otimes\boldsymbol{I}_n - \operatorname{diag}\left(\operatorname{vec}\left((\boldsymbol{V}\boldsymbol{U}^\top-\boldsymbol{F}^\top)\circledast_k(\boldsymbol{V}\boldsymbol{U}^\top)\right)\right)$$
$$- \begin{bmatrix} \operatorname{diag}\left(\boldsymbol{V}_{:,1}\odot((2\boldsymbol{V}\boldsymbol{U}^\top-\boldsymbol{F}^\top)\boldsymbol{U}_{:,1})\right) & \cdots & \operatorname{diag}\left(\boldsymbol{V}_{:,1}\odot((2\boldsymbol{V}\boldsymbol{U}^\top-\boldsymbol{F}^\top)\boldsymbol{U}_{:,k})\right) \\ \vdots & \ddots & \vdots \\ \operatorname{diag}\left(\boldsymbol{V}_{:,k}\odot((2\boldsymbol{V}\boldsymbol{U}^\top-\boldsymbol{F}^\top)\boldsymbol{U}_{:,1})\right) & \cdots & \operatorname{diag}\left(\boldsymbol{V}_{:,k}\odot((2\boldsymbol{V}\boldsymbol{U}^\top-\boldsymbol{F}^\top)\boldsymbol{U}_{:,k})\right) \end{bmatrix}$$

Then we have $\lambda_i(\nabla^2\mathcal{G}_S(\tilde{\boldsymbol{U}},\tilde{\boldsymbol{V}})) = \lambda_i(\nabla^2\mathcal{G}_S(\boldsymbol{U},\boldsymbol{V}))$ for all $i\in[(n+m)k] = \{1,2,\ldots,(n+m)k\}$ by noticing that we normalize the data as $\frac{\boldsymbol{U}_{i,j}}{\|\boldsymbol{U}_{i,:}\|_2}$ for all $i\in[n]$ and $\frac{\boldsymbol{V}_{i,j}}{\|\boldsymbol{V}_{i,:}\|_2}$ for all $i\in[m]$. This finishes the proof.

APPENDIX C

EXPERIMENTAL SETTINGS AND ADDITIONAL RESULT IN CHAPTER 4

C.1 Experimental Details

C.1.1 General Settings

The network architectures used in chapter 4 are elaborated in Table C.1. Due to the limitations of our GPU resources, we mostly conduct experiments based on plain CNN-9 and ResNet-32 for CIFAR and ResNet-18 for ImageNet. For CIFAR-10 and CIFAR-100, we use ADAM for all the networks including the baseline. For ImageNet-2012, we use the SGD with momentum 0.9 for all the networks. If not specified, we use the batch normalization by default for all the experiments on object recognition. For the experiments against adversarial attacks, we use the plain CNN-9. We do not use the batch normalization for the adversarial attack experiments. All the experiments are implemented using TensorFlow library. We use the same data augmentation protocol for CIFAR-10, CIFAR-100 and ImageNet-2012 as [12]. For initialization of DCNets and baselines, we follow [38]. For modified ResNet-18 in ImageNet, we use the same initialization as [10].

Since we are already using optimization tricks on $\|\boldsymbol{w}\|$, we propose to replace the orthonormal constraint in [12] with the proposed orthogonal constraint.

C.1.2 Details about FGSM and BIM Attacks

Recent studies show that neural networks are prone to adversarial attacks [95, 192, 193, 21]. One of the simplest attacks is FGSM [95], which computes the adversarial image $\tilde{x}$ of some input image x such that $\|x-\tilde{x}\|_\infty \leq \epsilon$. FGSM performs one single step gradient descent (with step size ϵ) to decrease the probability of the ground truth label. Formally, $\tilde{x} = x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$ where J is the loss function used to train the network, θ represents

Table C.1: Our CNN and ResNet architectures with different convolutional layers. Conv0.x, Conv1.x, Conv2.x, Conv3.x and Conv4.x denote convolution units that may contain multiple convolutional layers, and residual units are shown in double-column brackets. Conv1.x, Conv2.x and Conv3.x usually operate on different size feature maps. These networks are essentially similar to [61], but with different number of filters in each layer. The downsampling is performed by convolutions with a stride of 2. E.g., [3×3, 64]×4 denotes 4 cascaded convolution layers with 64 filters of size 3×3, and S2 denotes stride 2.

Layer	Plain CNN-9	CNN-9 (adversarial attacks)	ResNet-32 for CIFAR	Standard ResNet-18	Modified ResNet-18
Conv0.x	N/A	N/A	[3×3, 96]	[7×7, 64], S2 3×3, Max Pooling, S2	[7×7, 128], S2 3×3, Max Pooling, S2
Conv1.x	[3×3, 64]×3 2×2 Max Pooling, S2	[3×3, 32]×3 2×2 Max Pooling, S2	$\begin{bmatrix} 3\times 3, 96 \\ 3\times 3, 96 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times 3, 64 \\ 3\times 3, 64 \end{bmatrix} \times 2$	[3×3, 128]×1, S2 $\begin{bmatrix} 3\times 3, 128 \\ 3\times 3, 128 \end{bmatrix} \times 1$
Conv2.x	[3×3, 128]×3 2×2 Max Pooling, S2	[3×3, 64]×3 2×2 Max Pooling, S2	$\begin{bmatrix} 3\times 3, 192 \\ 3\times 3, 192 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times 3, 128 \\ 3\times 3, 128 \end{bmatrix} \times 2$	[3×3, 256]×1, S2 $\begin{bmatrix} 3\times 3, 256 \\ 3\times 3, 256 \end{bmatrix} \times 2$
Conv3.x	[3×3, 256]×3 2×2 Max Pooling, S2	[3×3, 128]×3 2×2 Max Pooling, S2	$\begin{bmatrix} 3\times 3, 384 \\ 3\times 3, 384 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times 3, 256 \\ 3\times 3, 256 \end{bmatrix} \times 2$	[3×3, 512]×1, S2 $\begin{bmatrix} 3\times 3, 512 \\ 3\times 3, 512 \end{bmatrix} \times 3$
Conv4.x	N/A	N/A	N/A	$\begin{bmatrix} 3\times 3, 512 \\ 3\times 3, 512 \end{bmatrix} \times 2$	[3×3, 1024]×1, S2
Final	512-dim fully connected	256-dim fully connected	Average Pooling		

the network parameters, x is the input image and y is the ground truth label associated with x. We compare our models and ResNet baseline on the performance on adversarial examples.

In addition, we evaluate the performance of DCNets and ResNet baselines on BIM (Basic Iterative Method) attack [96]. BIM runs certain number N of iterations of FGSM, with a smaller step size τ. In each iteration, the resulted perturbed image $\tilde{x}$ is clipped so that $\|x - \tilde{x}\|_\infty \leq \epsilon$.

We implement the experiments with Cleverhans [194]. In adversarial training using FGSM, $\epsilon = 8$ is used to generated the adversarial examples. In all the following adversarial attack experiments, we set $\epsilon = 8$, $\tau = 2$, $N = 20$. We report the accuracy on adversarial examples for both naturally trained models and adversarially trained models using FGSM. The network architecture is shown in Table C.1.

C.1.3 Details about the Black-box Attacks

[97] shows that adversarially trained models behave significantly different on adversarial examples trained on itself and transferred adversarial examples. As suggested by [97], we report the resistance of our models against black-box attacks with ResNet baseline model. Specifically, the adversarial examples are computed from a CNN baseline with the same architecture as the target models. The generated adversarial examples are then used to attack those target models. The architecture and the attack parameters are kept the same as in the white-box experiment.

C.2 Training and Implementation Details

Improved Learning of Operator Radius. To facilitate the learning of the operator radius ρ, we multiply the average norm of local patch $\boldsymbol{x}$ to ρ. Taking TanhConv as an example, we implement it using the following form:

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \tanh\big(\frac{\|\boldsymbol{x}\|}{\rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\}}\big) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{C.1}$$

where ρ is learnable and it is initialized by a constant 1. The reason we are multiplying the average norm of $\|\boldsymbol{x}\|$ to ρ comes from our empirical observation that lots of ρ in the middle layers stay unchanged and can not be updated effectively. Compared to the original formulation, (Equation C.1) essentially performs an normalization to $\|\boldsymbol{x}\|$ and make its mean become 1 (*i.e.*, $\mathbb{E}\big(\frac{\|\boldsymbol{x}\|}{\mathbb{E}\{\|\boldsymbol{x}\|\}}\big) = 1$). This is also the reason we initialize ρ with 1. The gradient of the magnitude function $h(\cdot)$ w.r.t ρ can be large enough such that ρ is updated effectively. Therefore, for all the decoupled operators that have a learnable non-zero operator radius ρ (*e.g.*, BallConv, TanhConv, SegConv, etc.), we will multiply $\mathbb{E}\{\|\boldsymbol{x}\|\}$ to ρ in order to facilitate its learning. In practice, we use the moving average to compute $\mathbb{E}\{\|\boldsymbol{x}\|\}$, similar to BN [44]. Note that, each kernel will preserve its independent patch norm mean $\mathbb{E}\{\|\boldsymbol{x}\|\}$. BallConv is implemented using

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \cdot \frac{\min(\|\boldsymbol{x}\|, \rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\})}{\rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\}} \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{C.2}$$

and SegConv is implemented using

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \begin{cases} \alpha \|\boldsymbol{x}\| \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), & 0 \leq \|\boldsymbol{x}\| \leq \rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\} \\ (\beta \|\boldsymbol{x}\| + \alpha\rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\} - \beta\rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\}) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), & \rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\} < \|\boldsymbol{x}\| \end{cases}. \tag{C.3}$$

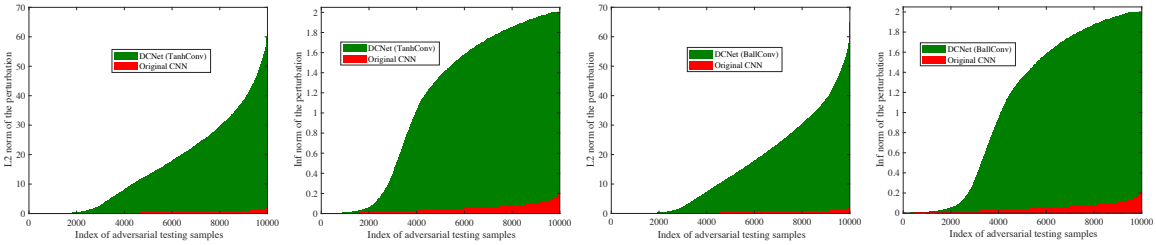
Hyperparameter Settings. For SphereConv, we use $\alpha = 1$. For BallConv, we use $\alpha = 1$ and a learnable ρ. For TanhConv, we use $\alpha = 1$ and a learnable ρ. For LinearConv, we use $\alpha = 1$. For SegConv, we use $\alpha = 1$, $\beta = 0.5$ and a learnable ρ. For LogConv, we use $\alpha = 1$. For MixConv, we use $\alpha = 1$ and $\beta = 1$. For CIFAR experiments, we use 128 batch size for all the networks. For ImageNet experiments, we use 40 batch size for all the networks.

C.3 More Experiments on Defense against Adversarial Attacks

We also evaluate the robustness of DCNets with the DeepFool attacks [193]. Note that, for all the experiments related to the adversarial attacks, our network do not use any optimization trick and is trained by original gradients. The results are given in Figure C.1. The x-axis denotes the index of the 10000 adversarial testing samples, and the y-axis denotes the strength (ℓ_2 norm or ℓ_∞ norm) of the perturbations in order to successfully fool the network. We could observe from the results that in order to fool the DCNets, the DeepFool attacks need to largely perturb the samples while it only takes a much smaller perturbation to fool the original CNNs. It implies that DCNets are much more difficult to fool. In other words, to fool the DCNets will take much more efforts than to fool the original CNNs, which shows the superior robustness of DCNets against adversarial examples.

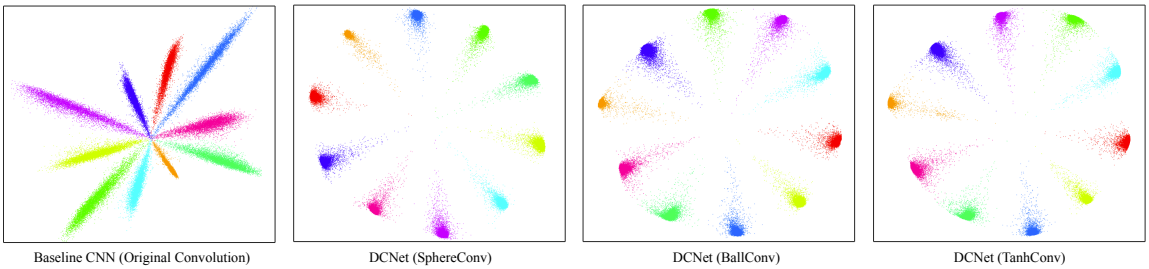
C.4 Feature Visualization on MNIST Dataset

We visualize the 2D feature on MNIST dataset. Specifically, we use a plain CNN with 6 convolutional layers ([3×3,32]×2-[3×3,64]×2-[3×3,128]×2) and 3 fully connected layers (256-2-10). Note that we set the output dimension (*i.e.*, the input dimension of the



(a) DCNet (TanhConv) vs. original CNN (b) DCNet (BallConv) vs. original CNN

Figure C.1: The strength of the adversarial perturbations to fool the network.



Baseline CNN (Original Convolution) DCNet (SphereConv) DCNet (BallConv) DCNet (TanhConv)

Figure C.2: 2D feature visualization on MNIST dataset with natural training.

last fully connected layer) as 2 and visualize these 2D features. We evaluate two types of training: natural training (*i.e.*, trained on the normal MNIST dataset) and adversarial training [95]. Note that, all the networks in this section are learned by original gradient updates. We do not use weight projection in the networks for the visualization purpose.

Natural Training. We plot the 2D features in Figure C.2. We could observe that DCNets (especially bounded decoupled operators) exhibit very different distributions with the original CNNs. Empirically, we observe that SphereConv, BallConv and TanhConv produce very compact and well-grouped features.

Adversarial Training. We also show the 2D features of adversarially trained models of baseline CNN, DCNet (SphereConv), DCNet (BallConv) and DCNet (TanhConv) in Figure C.3. We could see that DCNets are still able to group the features in a more compact way than the original CNN even with the adversarial training.

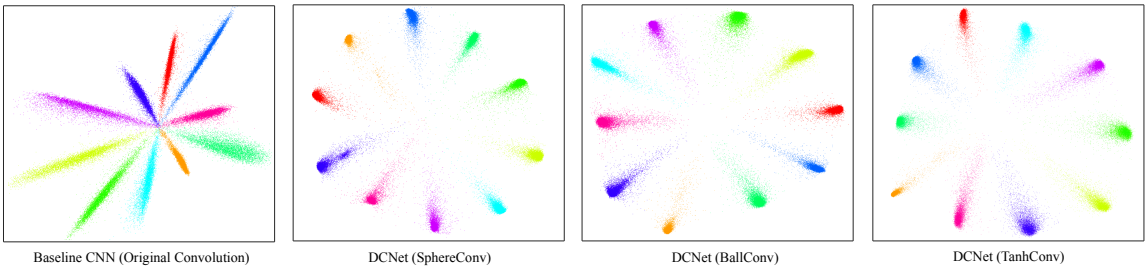


Figure C.3: 2D feature visualization on MNIST dataset with adversarial training.

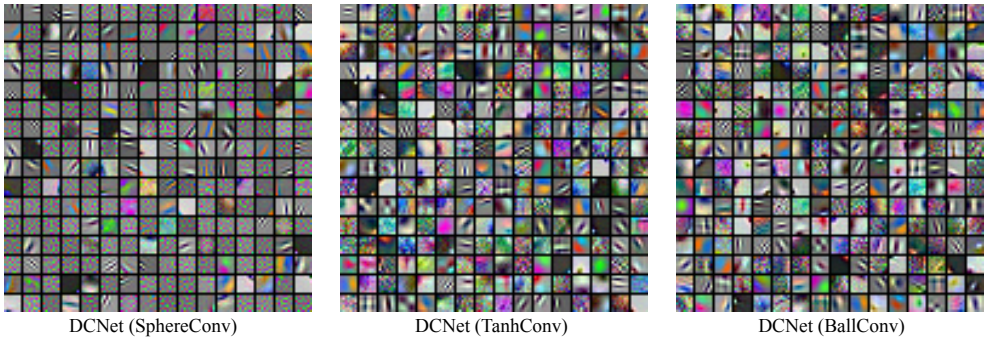


Figure C.4: Visualized filters from the first layer of DCNets on ImageNet-2012 dataset. Note that, this is learned by original gradient updates. We do not use weight projection in the networks for the visualization purpose.

C.5 Filter Visualization on ImageNet-2012

We train larger models with 256 filters in the first layer on ImageNet-2012. We visualize all the filters in the first layer for these compared methods in Figure C.4. One could see that DCNet with SphereConv learns more sparse filters, while DCNets with TanhConv and BallConv can learn richer types of filters. Moreover, because we use orthogonality constraints, the filters are not highly correlated unlike the original CNNs.

C.6 Experiments on CIFAR-100 (stochastic gradient descent)

Additionally, we optimize the DCNets with stochastic gradient descent (SGD) with momentum and evaluate our models on CIFAR-100. We use the ResNet-32 architecture. The experimental setting is the same as the CIFAR-100 experiment in chapter 4, except that we use SGD instead of ADAM. the results are given in Table C.2. Surprisingly, using

Table C.2: Testing error rate (%) of SGD-trained ResNet-32 on CIFAR-100.

Method	Linear	Cosine	Square Cosine
ResNet Baseline	-	21.55	-
SphereConv	21.71	21.61	24.62
BallConv	20.96	21.25	24.40
TanhConv	21.07	21.12	24.29
LinearConv	21.43	21.25	**20.54**
SegConv	**20.58**	**20.61**	20.61
LogConv	21.15	21.42	23.10
MixConv	20.82	21.20	21.19

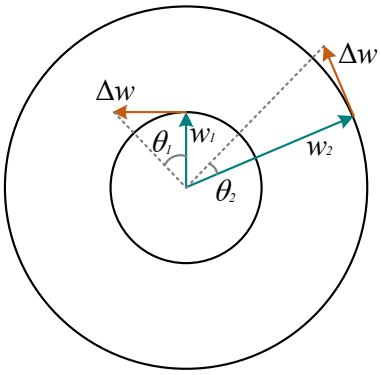


Figure C.5: Illustration of weight update, given fixed $||\Delta w||$. Notice that with $||w_1|| < ||w_2||$, $\theta_1 > \theta_2$.

SGD could largely improve the performance of baseline. While optimizing the baseline model using ADAM gives us 26.69% error rate, optimizing the baseline using SGD gives us 21.55% error rate. Even though the baseline will be greatly improved by SGD, we still find that our DCNets optimized by SGD are better than the baseline and also perform slightly better than the DCNets optimized by ADAM.

C.7 Difference between Weighted Gradients and Weight Projection

It is important to point out that the difference between weight projection and weighted gradients. Although weighted gradients eliminate the effect of normalizing w on the norm of gradients, the increment in angle (i.e. $\theta_{(w,x)}$) is different from that of weighted projection. Consider the simple case of LinearConv. Denote $y = \langle \frac{w}{||w||}, x \rangle$. Obviously, the gradient $||\nabla_w y||$ is always perpendicular to w. Therefore, the original gradient update is optimizing

the angle between w and x. The modified gradient update Δw is $\nabla_w y \cdot ||w||$ if using weighted gradients, and $\nabla_w y$ if using weight projection.

In the case of weighted gradients, even if all updates Δw have the same norm, the increment in 'angle' $\Delta\theta = \theta_{(w-\alpha\Delta w,x)} - \theta_{(w,x)}$ can vary, where α is the learning rate. Suppose the norm of the modified gradient $\|\Delta w\| = \|\nabla_w y\| \cdot \|w\|$ is fixed. $\Delta\theta$ can be ignored if $||w||$ is large, while close to 90 degrees if $\|w\|$ is extremely small. In other words, even if Δw is not dependent on $\|w\|$, $\Delta\theta$ is. See Figure C.5 for illustration.

In contrast, weighted projection forces the norm of the weights $\|\boldsymbol{w}\|$ to be a constant s, so when we have fixed gradient $\|\Delta w\|$, the change of angle $\Delta\theta$ will also be a constant (because $\boldsymbol{w}$ and $\Delta\boldsymbol{w}$ are always perpendicular to each other).

To summarize, weighted gradients make the update of angle $\Delta\theta$ more "adaptive", while weight projection makes the update of angle $\Delta\theta$ more "fixed". The major reason for such difference is that the norm of the weights $\|\boldsymbol{w}\|$ is fixed to a constant s in weight projection, while the norm of the weights $\|\boldsymbol{w}\|$ is not a constant in weighted gradients. Different $\|\boldsymbol{w}\|$ refers to a hypersphere with different radius, as shown in Figure C.5.

APPENDIX D

ADDITIONAL RESULTS AND PROOFS IN CHAPTER 5

D.1 Experimental Details

Table D.1: Our plain CNN architectures with different convolutional layers. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers. E.g., [3×3, 64]×3 denotes 3 cascaded convolution layers with 64 filters of size 3×3.

Layer	CNN-6	CNN-9	CNN-15
Conv1.x	[3×3, 64]×2	[3×3, 64]×3	[3×3, 64]×5
Pool1	2×2 Max Pooling, Stride 2		
Conv2.x	[3×3, 128]×2	[3×3, 128]×3	[3×3, 128]×5
Pool2	2×2 Max Pooling, Stride 2		
Conv3.x	[3×3, 256]×2	[3×3, 256]×3	[3×3, 256]×5
Pool3	2×2 Max Pooling, Stride 2		
Fully Connected	256	256	256

Table D.2: Our ResNet architectures with different convolutional layers. Conv0.x, Conv1.x, Conv2.x, Conv3.x and Conv4.x denote convolution units that may contain multiple convolutional layers, and residual units are shown in double-column brackets. Conv1.x, Conv2.x and Conv3.x usually operate on different size feature maps. These networks are essentially the same as [2], but some may have a different number of filters in each layer. The downsampling is performed by convolutions with a stride of 2. E.g., [3×3, 64]×4 denotes 4 cascaded convolution layers with 64 filters of size 3×3, and S2 denotes stride 2.

Layer	ResNet-32 for CIFAR-10/100	ResNet-18 for ImageNet-2012	ResNet-34 for ImageNet-2012
Conv0.x	N/A	[7×7, 64], Stride 2 3×3, Max Pooling, Stride 2	[7×7, 64], Stride 2 3×3, Max Pooling, Stride 2
Conv1.x	$[3\times3, 64]\times1$ $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times5$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$
Conv2.x	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times5$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$
Conv3.x	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times5$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$
Conv4.x	N/A	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$
	Average Pooling		

General settings. The network architectures used in the paper are elaborated in Ta-

ble D.1 Table D.2. For CIFAR-10 and CIFAR-100, we use batch size 128. We start with learning rate 0.1, divide it by 10 at 20k, 30k and 37.5k iterations, and terminate training at 42.5k iterations. For ImageNet-2012, we use batch size 64 and start with learning rate 0.1. The learning rate is divided by 10 at 150k, 300k and 400k iterations, and the training is terminated at 500k iterations. Note that, for all the compared methods, we always use the best possible hyperparameters to make sure that the comparison is fair. The baseline has exactly the same architecture and training settings as the one that MHE uses, and the only difference is an additional MHE regularization. For full-space MHE in hidden layers, we set λ_{h} as 10 for all experiments. For half-space MHE in hidden layers, we set λ_{h} as 1 for all experiments. For MHE in output layers, we set λ_{o} as 1 for all experiments. We use $1e-5$ for the orthonormal regularization. If not otherwise specified, standard ℓ_2 weight decay ($1e-4$) is applied to all the neural network including baselines and the networks that use MHE regularization. A very minor issue for the hyperparameters λ_{h} is that it may increase as the number of layers increases, so we can potentially further divide the hyperspherical energy for the hidden layers by the number of layers. It will probably change the current optimal hyperparameter setting by a constant multiplier. For notation simplicity, we do not explicitly write out the weight decay term in the loss function in the main paper. Note that, all the neuron weights in the neural networks used in the paper are not normalized (unless otherwise specified), but the MHE will normalize the neuron weights while computing the regularization loss. As a result, *MHE does not need to modify any component of the original neural networks, and it can simply be viewed as an extra regularization loss that can boost the performance*. Because half-space variants can only applied to the hidden layers, both original MHE and its half-space version apply the full-space MHE to the output layer by default. The difference between MHE and half-space MHE are only in the regularization for the hidden layers.

Class-imbalance learning. There are 50000 training images in the original CIFAR-10 dataset, with 5000 images per class. For the single class imbalance setting, we keep original

images of class 1-9 and randomly throw away 90% images of class 0. The total number of training images in this setting is 45500. For the multiple class imbalance setting, we set the number of each class equals to $500 \times (\text{class\_index} + 1)$. For instance, class 0 has 500 images, class 1 has 1000 images and class 9 has 5000 images. The total number of training images in this setting is 27500. Note that, both half-space MHE and half-space A-MHE in Table 5.7 and Figure D.4 mean that the half-space variants have been applied to the hidden layers. For the output layer (*i.e.*, classifier neurons), only full-space MHE can be used.

SphereFace+. SphereFace+ uses the same face detection and alignment method [73] as SphereFace [10]. The testing protocol on LFW and MegaFace is also the same as SphereFace. We use exactly the same preprocessing as in the SphereFace repository. Detailed network architecture settings of SphereFace-20 and SphereFace-64 can be found in [10]. Specifically, we use full-space MHE with Euclidean distance and $s = 2$ in the output layer. Essentially, we treat MHE as an additional loss function which aims to enlarge the inter-class angular distance of features and serves a complementary role to the angular softmax in SphereFace. Note that, for the results of CosineFace [13], we directly use the results (with the same training settings and without using feature normalization) reported in the paper. Since ours also does not perform feature normalization, it is a fair comparison. With feature normalization, we find that the performance of SphereFace+ will also be improved significantly. However, feature normalization makes the results more tricky, because it will involve another hyperparameter that controls the projection radius of feature normalization.

In order to reduce the training difficulty, we adopt a new training strategy. Specifically, we first train a model using the original SphereFace, and then use the new loss function proposed in Equation 5.8 to finetune the pretrained SphereFace model. Note that, only the results for face recognition are obtained using this training strategy.

D.2 Proof of Theorem 1 and Theorem 2

Theorem 1 and Theorem 2 are natural results from classic potential theory [123] and spherical configuration [124, 121, 125]. We discuss the asymptotic behavior ($N \to \infty$) in three cases: $0 < s < d$, $s = d$, and $s > d$. We first write the energy integral as

$$I_s(\mu) = \iint_{\mathbb{S}^d \times \mathbb{S}^d} \|\boldsymbol{u} - \boldsymbol{v}\|^{-s} d\mu(\boldsymbol{u}) d\mu(\boldsymbol{v}), \tag{D.1}$$

which is taken over all probability measure μ supported on $\mathbb{S}^d$. With $0 < s < d$, $I_s(\mu)$ is minimal when μ is the spherical measure $\sigma^d = \mathcal{H}^d(\cdot)|_{\mathbb{S}^d} / \mathcal{H}^d(\mathbb{S}^d)$ on $\mathbb{S}^d$, where $\mathcal{H}^d(\cdot)$ denotes the d-dimensional Hausdorff measure. When $s \geq d$, $I_s(\mu)$ becomes infinity, which therefore requires different analysis.

First, the classic potential theory [123] can directly give the following results for the case where $0 < s < d$:

Lemma 4. *If* $0 < s < d$*,*

$$\lim_{N \to \infty} \frac{\boldsymbol{\varepsilon}_{s,d}(N)}{N^2} = I_s\left(\frac{\mathcal{H}^d(\cdot)|_{\mathbb{S}^d}}{\mathcal{H}^d(\mathbb{S}^d)}\right), \tag{D.2}$$

where I_s *is defined in the main paper. Moreover, any sequence of optimal hyperspherical* s*-energy configurations* $(\hat{\boldsymbol{W}}_N^\star)|_2^\infty \subset \mathbb{S}^d$ *is asymptotically uniformly distributed in the sense that for the weak-star topology measures,*

$$\frac{1}{N} \sum_{\boldsymbol{v} \in \hat{\boldsymbol{W}}_N^\star} \delta_{\boldsymbol{v}} \to \sigma^d, \qquad \text{as } N \to \infty \tag{D.3}$$

where $\delta_{\boldsymbol{v}}$ *denotes the unit point mass at* $\boldsymbol{v}$*, and* σ^d *is the spherical measure on* $\mathbb{S}^d$*.*

which directly concludes Theorem 1 and Theorem 2 in the case of $0 < s < d$.

For the case where $s = d$, we have from [121, 125] the following results:

Lemma 5. *Let $\mathcal{B}^d := \bar{B}(0,1)$ be the closed unit ball in $\mathbb{R}^d$. For $s = d$,*

$$\lim_{N\to\infty} \frac{\boldsymbol{\varepsilon}_{s,d}(N)}{N^2 \log N} = \frac{\mathcal{H}^d(\mathcal{B}^d)}{\mathcal{H}^d(\mathbb{S}^d)} = \frac{1}{d}\frac{\Gamma(\frac{d+1}{2})}{\sqrt{\pi}\Gamma(\frac{d}{2})}, \tag{D.4}$$

and any sequence $(\hat{\boldsymbol{W}}_N^\star)|_2^\infty \subset \mathbb{S}^d$ of optimal s-energy configurations satisfies Equation D.3.

which concludes the case of $s = d$. Therefore, we are left with the case where $s > d$. For this case, we can use the results from [124]:

Lemma 6. *Let $A \subset \mathbb{R}^d$ be compact with $\mathcal{H}_d(A) > 0$, and $\tilde{W}_N = \{x_{k,N}\}_{i=1}^N$ be a sequence of asymptotically optimal N-point configurations in A in the sense that for some $s > d$,*

$$\lim_{N\to\infty} \frac{\boldsymbol{E}_s(\tilde{W}_N)}{N^{1+s/d}} = \frac{C_{s,d}}{\mathcal{H}^d(A)^{s/d}} \tag{D.5}$$

or

$$\lim_{N\to\infty} \frac{\boldsymbol{E}_s(\tilde{W}_N)}{N^2 \log N} = \frac{\mathcal{H}^d(\mathcal{B}^d)}{\mathcal{H}^d(A)}. \tag{D.6}$$

where $C_{s,d}$ is a finite positive constant independent of A. Let δ_x be the unit point mass at the point x. Then in the weak-star topology of measures we have

$$\frac{1}{N}\sum_{i=1}^{N} \delta_{x_{i,N}} \to \frac{\mathcal{H}^d(\cdot)|_A}{\mathcal{H}^d(A)}, \quad asN \to \infty. \tag{D.7}$$

The results naturally prove the case of $s > d$. Combining these three lemmas, we have proved Theorem 1 and Theorem 2.

D.3 Understanding MHE from Decoupled View

Inspired by decoupled networks [20], we can view the original convolution as the multiplication of the angular function $g(\theta) = \cos(\theta)$ and the magnitude function $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) =$

$\|\boldsymbol{w}\| \cdot \|\boldsymbol{x}\|$:

$$\begin{aligned} f(\boldsymbol{w}, \boldsymbol{x}) &= h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) \cdot g(\theta) \\ &= \big(\|\boldsymbol{w}\| \cdot \|\boldsymbol{x}\|\big) \cdot \big(\cos(\theta)\big) \end{aligned} \tag{D.8}$$

where θ is the angle between the kernel $\boldsymbol{w}$ and the input $\boldsymbol{x}$. From the equation above, we can see that the norm of the kernel and the direction (*i.e.*, angle) of the kernel affect the inner product similarity differently. Typically, weight decay is to regularize the kernel by minimizing its ℓ_2 norm, while there is no regularization on the direction of the kernel. Therefore, MHE is able to complete this missing piece by promoting angular diversity. By combining MHE to a standard neural networks (*e.g.*, CNNs), the regularization term becomes

$$\begin{aligned} \mathcal{L}_{\text{reg}} = \quad & \underbrace{\lambda_{\text{w}} \cdot \frac{1}{\sum_{j=1}^{L} N_j} \sum_{j=1}^{L} \sum_{i=1}^{N_j} \|\boldsymbol{w}_i\|}_{\text{Weight decay: regularizing the magnitude of kernels}} \\ & \underbrace{+ \lambda_{\text{h}} \cdot \sum_{j=1}^{L-1} \frac{1}{N_j(N_j-1)} \{\boldsymbol{E}_s\}_j + \lambda_{\text{o}} \cdot \frac{1}{N_L(N_L-1)} \boldsymbol{E}_s(\hat{\boldsymbol{w}}_i^{\text{out}}|_{i=1}^{c})}_{\text{MHE: regularizing the direction of kernels}} \end{aligned} \tag{D.9}$$

where $\boldsymbol{x}_i$ is the feature of the i-th training sample entering the output layer, $\boldsymbol{w}_i^{\text{out}}$ is the classifier neuron for the i-th class in the output fully-connected layer and $\hat{\boldsymbol{w}}_i^{\text{out}}$ denotes its normalized version. $\{\boldsymbol{E}_s\}_i$ denotes the hyperspherical energy for the neurons in the i-th layer. c is the number of classes, m is the batch size, L is the number of layers of the neural network, and N_i is the number of neurons in the i-th layer. $\boldsymbol{E}_s(\hat{\boldsymbol{w}}_i^{\text{out}}|_{i=1}^{c})$ denotes the hyperspherical energy of neurons $\{\hat{\boldsymbol{w}}_1^{\text{out}}, \cdots, \hat{\boldsymbol{w}}_c^{\text{out}}\}$ in the output layer. λ_{w}, λ_{h} and λ_{o} are weighting hyperparameters for these three regularization terms.

From the decoupled view, we can see that MHE is actually very meaningful in regularizing the neural networks, and it also serves as a complementary role to weight decay. According to [20] (using classifier neurons as an intuitive example), weight decay is used to regularize the intra-class variation, while MHE is used to regularize the inter-class seman-

tic difference. In such sense, MHE completes an important missing piece for the standard neural networks by regularizing the directions of neurons (*i.e.*, kernels). In contrast, the standard neural networks only have weight decay as a regularization for the norm of neurons.

Weight decay can help to prevent the network from overfitting and improve the generalization. Similarly, MHE can serve as a similar role, and we argue that MHE is very likely to be more crucial than weight decay in avoiding overfitting and improving generalization. Our intuition comes from SphereNets [12] which shows that the magnitude of kernels is not important for object recognition. Therefore, the directions of the kernels are directly related to the semantic discrimination of the neural networks, and MHE is designed to regularize the directions of kernels by imposing the hyperspherical diversity. To conclude, MHE provides a novel hyperspherical perspective for regularizing neural networks.

D.4 Weighted MHE

In this section, we do a preliminary study for weighted MHE. To be clear, weighted MHE is to compute MHE with neurons with different fixed weights. Taking Euclidean distance MHE as an example, weighted MHE can be formulated as:

$$\begin{aligned}\boldsymbol{E}_{s,d}(\beta_i\hat{\boldsymbol{w}}_i|_{i=1}^N) &= \sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s\big(\|\beta_i\hat{\boldsymbol{w}}_i-\beta_j\hat{\boldsymbol{w}}_j\|\big)\\ &= \begin{cases}\sum_{i\neq j}\|\beta_i\hat{\boldsymbol{w}}_i-\beta_j\hat{\boldsymbol{w}}_j\|^{-s}, & s>0\\ \sum_{i\neq j}\log\big(\|\beta_i\hat{\boldsymbol{w}}_i-\beta_j\hat{\boldsymbol{w}}_j\|^{-1}\big), & s=0\end{cases},\end{aligned} \tag{D.10}$$

where β_i is a constant weight for the neuron $\boldsymbol{w}_i$. We perform a toy experiment to see how these weights β_i can affect the neuron distribution on 3-dimensional sphere. Specifically, we follow the same setting as Figure 5.1, and apply weighted MHE to 10 normalized vectors in 3-dimensional space. We experiment two settings: (1) only one neuron $\boldsymbol{w}_1$ has different weight β_1 than the other 9 neurons; (2) two neurons $\boldsymbol{w}_1, \boldsymbol{w}_2$ have different

weight β_1, β_2 than the other 8 neurons. For the first setting, we visualize the cases where $\beta_1 = 1, 2, 4, 10$ and $\beta_i = 1, 10 \geq i \geq 2$. The visualization results are shown in Figure D.1. For the second setting, we visualize the cases where $\beta_1 = \beta_2 = 1, 2, 4, 10$ and $\beta_i = 1, 10 \geq i \geq 3$. The visualization results are shown in Figure D.2. In these visualization experiments, we only use the gradient of weighted MHE to update the randomly initialized neurons. Note that, for all experiments, the random seed is fixed.

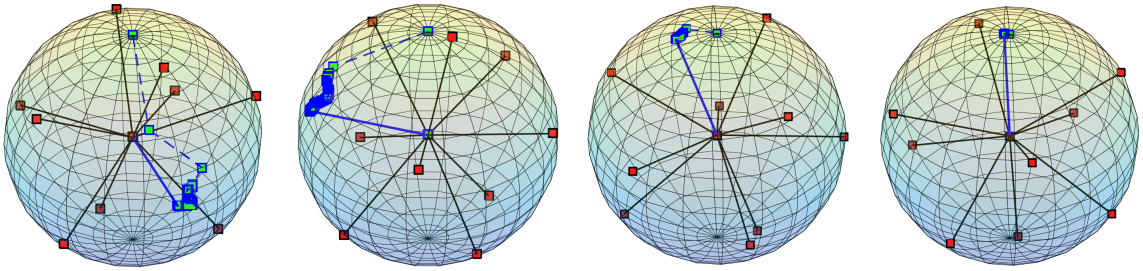


Figure D.1: The visualization of normalized neurons after applying weighted MHE in the first setting. The blue-green square dots denote the trajectory (history of the iterates) of neuron $\boldsymbol{w}_1$ with $\beta_1 = 1, 2, 4, 10$, while the red dots denote the neurons with $\beta_i = 1, i \neq 1$. The final neuron $\boldsymbol{w}_1$ is connected to the origin with a solid blue line. The dash line is used to connected the trajectory.

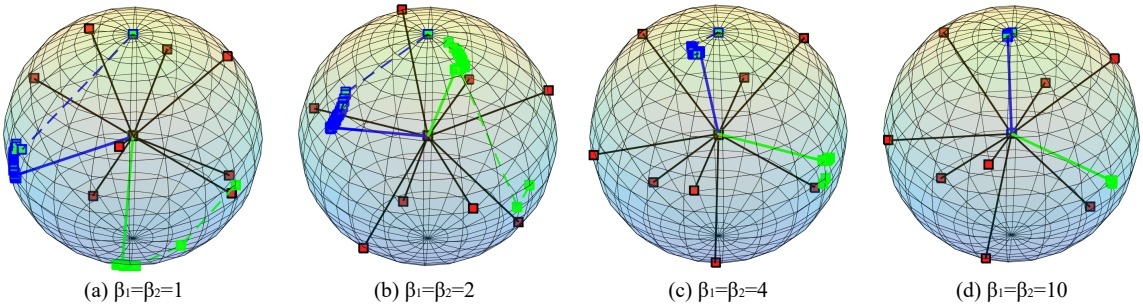


Figure D.2: The visualization of normalized neurons after applying weighted MHE in the second setting. The blue-green square dots denote the trajectory of neuron $\boldsymbol{w}_1$ with $\beta_1 = 1, 2, 4, 10$, the pure green square dots denote the trajectory of neuron $\boldsymbol{w}_2$ with $\beta_2 = 1, 2, 4, 10$, and the red dots denote the neurons with $\beta_i = 1, i \neq 1, 2$. The final neurons $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are connected to the origin with a solid blue line and a solid green line, respectively. The dash line is used to connected the trajectory.

From both Figure D.1 and Figure D.2, one can observe that the neurons with larger β tend to be more "fixed" (unlikely to move), and the neurons with smaller β tend to

move more flexibly. This can also be interpreted as the neurons with larger β being more important. Such phenomena indicate that we can control the flexibility of the neurons under the learning dynamics of MHE. There is one scenario where weighted MHE may be very useful. Suppose we have known that some neurons are already well learned (*e.g.*, some filters from a pretrained model) and we do not want these neurons to be updated dramatically, then we can use the weighted MHE and set a larger β for these neurons.

D.5 Regularizing SphereNets with MHE

SphereNets [12] are a family of network networks that learns on hyperspheres. The filters in SphereNets only focus on the hyperspherical (*i.e.*, angular) difference. One can see that the intuition of SphereNets well matches that of MHE, so MHE can serve as a natural and effective regularization for SphereNets. Because SphereNets throw away all the magnitude information of filters, the weight decay can no longer serve as a form of regularization for SphereNets, which makes MHE a very useful regularization for SphereNets. Originally, we use the orthonormal regularization $\|\boldsymbol{W}^\top\boldsymbol{W}-\boldsymbol{I}\|_F^2$ to regularize SphereNets, where $\boldsymbol{W}$ is the weight matrix of a layer with each column being a vectorized filter and $\boldsymbol{I}$ is an identity matrix. We compare MHE, half-space MHE and orthonormal regularization for SphereNets. In this section, all the SphereNets use the same architecture as the CNN-9 in Table D.1, the training setting is also the same as CNN-9. We only evaluate SphereNets with cosine SphereConv. Note that, $s = 0$ is actually the logarithmic hyperspherical energy (a relaxation of the original hyperspherical energy). From Table D.3, we observe that SphereNets with MHE can outperform both the SphereNet baseline and SphereNets with the orthonormal regularization, showing that MHE is not only effective in standard CNNs but also very suitable for SphereNets.

Table D.3: Testing error (%) of SphereNet with different MHE on CIFAR-10/100.

Method	CIFAR-10			CIFAR-100		
	$s=2$	$s=1$	$s=0$	$s=2$	$s=1$	$s=0$
MHE	**5.71**	5.99	**5.95**	27.28	**26.99**	27.03
Half-space MHE	6.12	6.33	6.31	27.17	27.77	27.46
A-MHE	5.91	5.98	6.06	**27.07**	27.27	**26.70**
Half-space A-MHE	6.14	**5.87**	6.11	27.35	27.68	27.58
SphereNet with Orthonormal Reg.	6.13			27.95		
SphereNet Baseline	6.37			28.10		

D.6 Improving AM-Softmax with MHE

We also perform some preliminary experiments for applying MHE to additive margin softmax loss [14] which is a recently proposed well-performing objective function for face recognition. The loss function of AM-Softmax is given as follows:

$$\mathcal{L}_{\text{AMS}} = -\frac{1}{n}\sum_{i=1}^{n}\log\frac{e^{s\cdot\left(cos\theta_{(\boldsymbol{x}_i,\boldsymbol{w}_{y_i})}-m_{\text{AMS}}\right)}}{e^{s\cdot\left(cos\theta_{(\boldsymbol{x}_i,\boldsymbol{w}_{y_i})}-m_{\text{AMS}}\right)}+\sum_{j=1,j\neq y_i}^{c}e^{s\cdot cos\theta_{(\boldsymbol{x}_i,\boldsymbol{w}_j)}}} \tag{D.11}$$

where y_i is the label of the training sample x_i, n is the mini-batch size, m_{AMS} is the hyperparameter that controls the degree of angular margin, and $\theta_{(\boldsymbol{x}_i,\boldsymbol{w}_j)}$ denotes the angle between the training sample $\boldsymbol{x}_i$ and the classifier neuron $\boldsymbol{w}_j$. s is the hyperparameter that controls the projection radius of feature normalization [11, 14]. Similar to our SphereFace+, we combine full-space MHE to the output layer to improve the inter-class feature separability. It is essentially following the same intuition of SphereFace+ by adding an additional loss function to AM-Softmax loss.

Experiments. We perform a preliminary experiment to study the benefits of MHE for improving AM-Softmax loss. We use the SphereFace-20 network and trained on CASIA-WebFace dataset (training settings are exactly the same as SphereFace+ in the main paper and [10]). The hyperparameters s, m_{AMS} for AM-Softmax loss exactly follow the best setting in [14]. AM-Softmax achieves 99.26% accuracy on LFW, while combining MHE with AM-Softmax yields 99.37% accuracy on LFW. Such performance gain is actually very significant in face verification, which further validates the superiority of MHE.

D.7 Improving GANs with MHE

We propose to improve the discriminator of GANs using MHE. It has been pointed out in [118] that the function space from which the discriminators are learned largely affects the performance of GANs. Therefore, it is of great importance to learn a good discriminator for GANs. As a recently proposed regularization to stabilize the training of GANs, spectral normalization (SN) [118] encourages the Lipschitz constant of each layer's weight matrix to be one. Since MHE exhibits significant performance gain for CNNs as a regularization, we expect MHE can also improve the training of GANs by regularizing its discriminator. As a result, we perform a preliminary evaluation on applying MHE to GANs.

Specifically, for all methods except WGAN-GP [195], we use the standard objective function for the adversarial loss:

$$V(G, D) := \mathbb{E}_{\boldsymbol{x}\sim q_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))], \tag{D.12}$$

where $\boldsymbol{z} \in \mathbb{R}^{d_z}$ is a latent variable, $p(\boldsymbol{z})$ is the normal distribution $\mathcal{N}(0, I)$, and $G : \mathbb{R}^{d_z} \to \mathbb{R}^{d_0}$ is a deterministic generator function. We set d_z to 128 in all the experiments. For the updates of G, we used the alternate cost proposed by [143] $-\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(D(G(\boldsymbol{z})))]$ as used in [143, 196]. For the updates of D, we used the original cost function defined in Equation D.12.

Recall from [118] that spectral normalization normalizes the spectral norm of the weight matrix $\boldsymbol{W}$ such that it makes the Lipschitz constraint $\sigma(\boldsymbol{W})$ to be one:

$$\bar{\boldsymbol{W}}_{\text{SN}}(\boldsymbol{W}) := \frac{\boldsymbol{W}}{\sigma(\boldsymbol{W})}. \tag{D.13}$$

We apply MHE to the discriminator of standard GANs (with the original loss function in [143]) for image generation on CIFAR-10. In general, our experimental settings and training strategies (including architectures in Table D.5) exactly follow spectral normaliza-

tion [118]. For MHE, we use the half-space variant with Euclidean distance (Equation 5.1). We first experiment regularizing the discriminator using MHE alone, and it yields comparable performance to SN and orthonormal regularization. Moreover, we also regularize the discriminator simultaneously using both MHE and SN, and it can give much better results than using either SN or MHE alone. The results in Table D.4 show that MHE is potentially very useful for training GANs.

Table D.4: Inception scores with unsupervised image generation on CIFAR-10.

Method	Inception score
Real data	11.24±.12
Weight clipping	6.41±.11
GAN-gradient penalty (GP)	6.93±.08
WGAN-GP [195]	6.68±.06
Batch Normalization [44]	6.27±.10
Layer Normalization [197]	7.19±.12
Weight Normalization [128]	6.84±.07
Orthonormal [117]	7.40±.12
SN-GANs [118]	**7.42**±.08
MHE (ours)	7.32±.10
MHE + SN [118] (ours)	**7.59**±.08

D.7.1 Network Architecture for GAN

We give the detailed network architectures in Table D.5 that are used in our experiments for the generator and the discriminator.

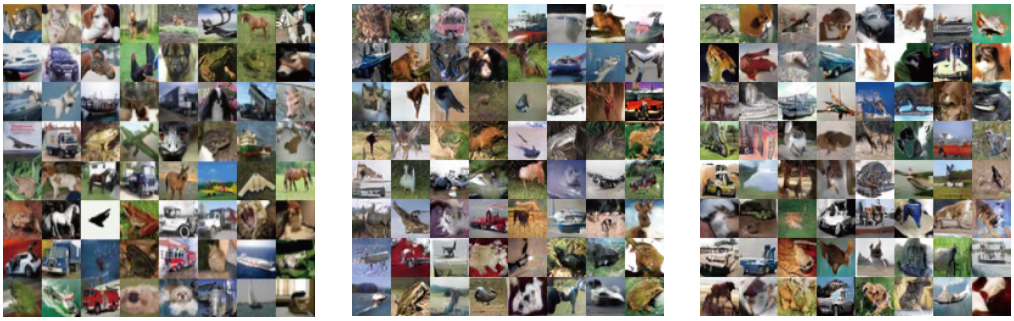
Table D.5: Our CNN architectures for image Generation on CIFAR-10. The slopes of all leaky ReLU (lReLU) functions in the networks are set to 0.1.

(a) Generator ($M_g = 4$ for CIFAR10).

$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
dense $\rightarrow M_g \times M_g \times 512$
4×4, stride=2 deconv. BN 256 ReLU
4×4, stride=2 deconv. BN 128 ReLU
4×4, stride=2 deconv. BN 64 ReLU
3×3, stride=1 conv. 3 Tanh

(b) Discriminator ($M = 32$ CIFAR10).

RGB image $x \in \mathbb{R}^{M \times M \times 3}$
3×3, stride=1 conv 64 lReLU 4×4, stride=2 conv 64 lReLU
3×3, stride=1 conv 128 lReLU 4×4, stride=2 conv 128 lReLU
3×3, stride=1 conv 256 lReLU 4×4, stride=2 conv 256 lReLU
3×3, stride=1 conv. 512 lReLU
dense $\rightarrow$ 1



Dataset | Baseline GAN | GAN with MHE and SN

Figure D.3: Results of generated images.

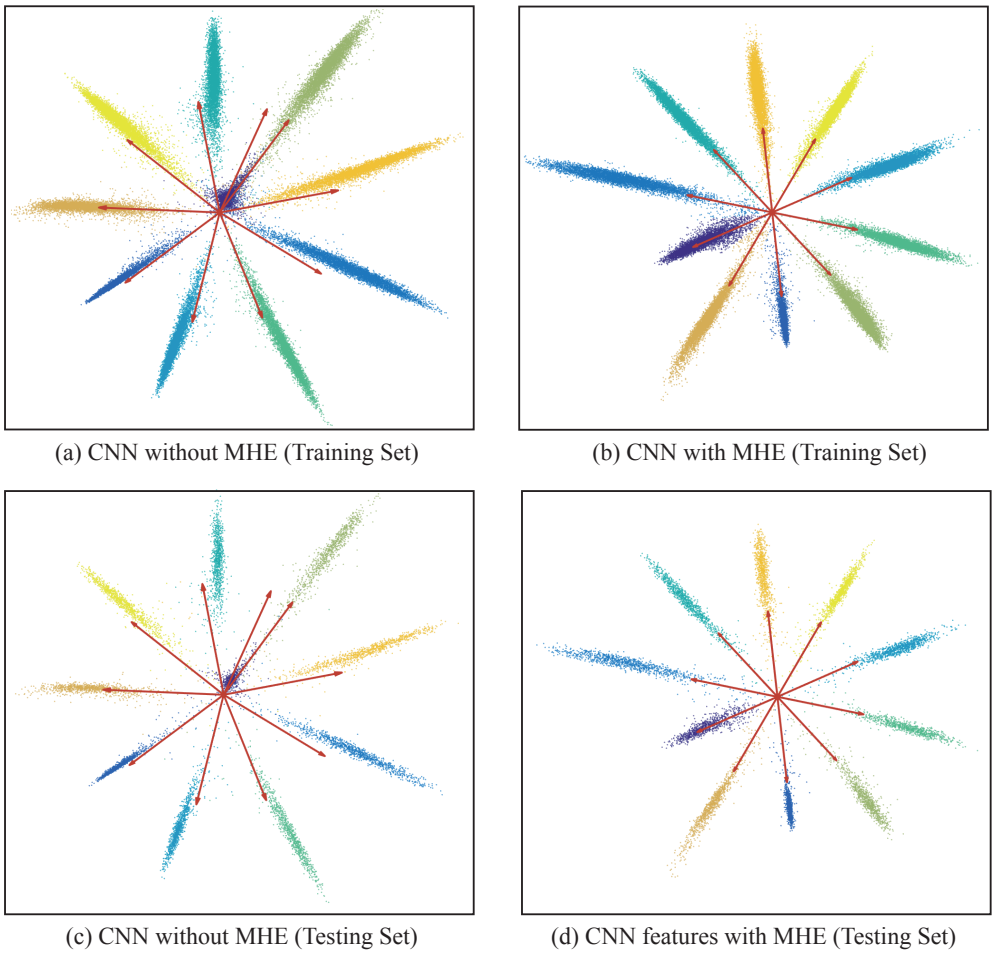
D.7.2 Comparison of Random Generated Images

We provide some randomly generated images for comparison between baseline GAN and GAN regularized by both MHE and SN. The generated images are shown in Figure D.3.

D.8 More Results on Class-imbalance Learning

D.8.1 Class-imbalance learning on CIFAR-100

We perform additional experiments on CIFAR-100 to further validate the effectiveness of MHE in class-imbalance learning. In the CNN used in the experiment, we only apply MHE (*i.e.*, full-space MHE) to the output layer, and use MHE or half-space MHE in the hidden layers. In general, the experimental settings are the same as the main paper. We still use CNN-9 (which is a 9-layer CNN from Table D.1) in the experiment. Slightly differently from CIFAR-10 in the main paper, the two data imbalance settings on CIFAR-100 include 1) 10-class imbalance (denoted as Single in Table D.6) - All classes have the same number of images but 10 classes (index from 0 to 9) have significantly less number (only 10% training samples compared to the other normal classes), and 2) multiple class imbalance (denoted by Multiple in Table D.6) - The number of images decreases as the class index decreases from 99 to 0. For the multiple class imbalance setting, we set the number of each class equals to $5 \times (\text{class\_index} + 1)$. Experiment details are similar to the CIFAR-10



(a) CNN without MHE (Training Set)

(b) CNN with MHE (Training Set)

(c) CNN without MHE (Testing Set)

(d) CNN features with MHE (Testing Set)

Figure D.4: 2D CNN features with or without MHE on both training set and testing set. The features are computed by setting the output feature dimension as 2, similar to [9]. Each point denotes the 2D feature of a data point, and each color denotes a class. The red arrows are the classifier neurons of the output layer.

experiment, which is specified in Appendix section D.1. The results in Table D.6 show that MHE consistently improves CNNs in class-imbalance learning on CIFAR-100. In most cases, half-space MHE performs better than full-space MHE.

D.8.2 2D CNN Feature Visualization

The experimental settings are the same as the main paper. We supplement the 2D feature visualization on testing set in Figure D.4. The visualized features on both training set and testing set well demonstrate the superiority of MHE in class-imbalance learning.

Table D.6: Error rate (%) on imbalanced CIFAR-100.

Method	Single	Multiple
Baseline	31.43	38.39
Orthonormal	30.75	37.89
MHE	**29.30**	37.07
Half-space MHE	29.40	**36.52**
A-MHE	30.16	37.54
Half-space A-MHE	29.60	37.07

In the CNN without MHE, the classifier neuron of the imbalanced training data is highly biased towards another class, and therefore can not be properly learned. In contrast, the CNN with MHE can learn uniformly distributed classifier neurons, which greatly improves the network's generalization ability.

Table D.7: Megaface Verification Rate (%) of SphereFace+ under Res-20

m_{SF}	SphereFace	SphereFace+
1	42.46	**52.02**
2	71.79	**80.94**
3	76.34	**80.58**
4	82.56	**83.39**

Table D.8: Performance of SphereFace+ trained on different datasets.

Dataset	# images	# identities	Method	LFW (%)	MegaFace (%)
IMDb-Face	1.7M	56K	SphereFace	99.53	72.89
			SphereFace+	**99.57**	**73.15**
MS-Celeb	8.6M	96K	SphereFace	99.48	73.93
			SphereFace+	**99.5**	**74.16**
CASIA-WebFace	0.49M	10.5K	SphereFace	99.27	70.68
			SphereFace+	**99.32**	**71.30**

D.9 More results of SphereFace+ on Megaface Challenge

We give more experimental results of SphereFace+ on Megaface challenge. The results in Table D.7 evaluate SphereFace+ under different m_{SF} and show that SphereFace+ consistently outperforms the SphereFace baseline. It indicates that MHE also enhances the verification rate on Megaface challenge. Our results of Identification Rate vs. Distractors Size and ROC curve are showed in Figure D.5 and Figure D.6, respectively.

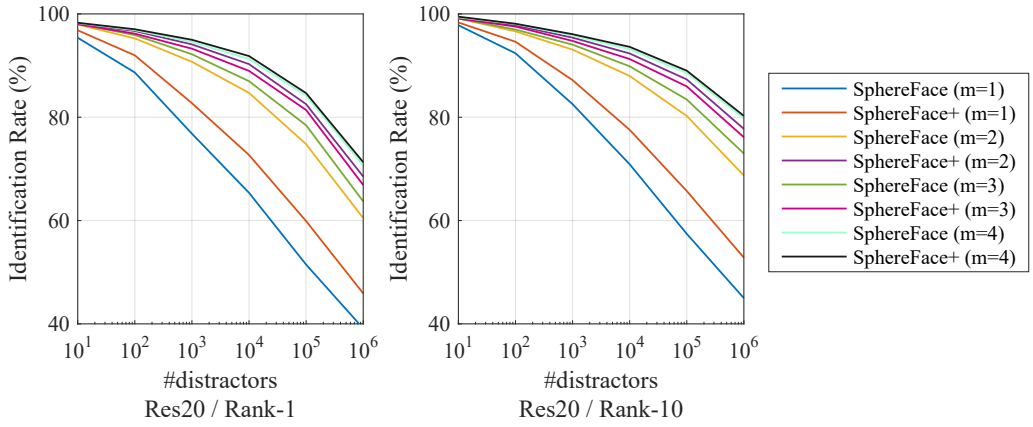


Figure D.5: Rank-1/Rank-10 Identification Performance on Megaface.

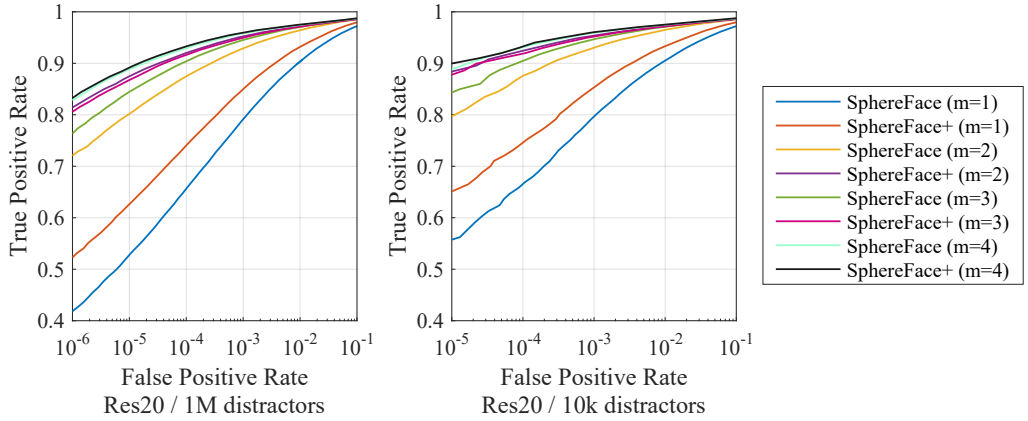


Figure D.6: ROC Curve with 1M/10k Distractors on Megaface.

D.10 Training SphereFace+ on MS-Celeb and IMDb-Face datasets

All the previous results of SphereFace+ are trained on the relatively small-scale CASIA-WebFace dataset which only has 0.49M images. In order to comprehensively evaluate SphereFace+, we further train the SphereFace+ model on two much larger datasets: MS-Celeb [198] (8.6M images) and IMDb-Face [199] (1.7M images). Specifically, we use the SphereFace-20 network architecture [10] for both SphereFace and SphereFace+ We evaluate SphereFace+ on both LFW (verification accuracy) and MegaFace (rank-1 identification accuracy with 1M distractors), and the results are given in Table D.8. From Table D.8, we can observe that SphereFace+ still consistently outperforms SphereFace with a noticeable margin. Note that, our SphereFace performance may differ from the ones in [199], because

we use different face detection and alignment tools. Most importantly, the results in Table D.8 are fairly compared, since all the preprocessings and network architectures used here are the same.

APPENDIX E

ADDITIONAL RESULTS AND PROOFS IN CHAPTER 6

E.1 Experimental Details

Table E.1: Our plain CNN architectures with different convolutional layers. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers. E.g., [3×3, 64]×3 denotes 3 cascaded convolution layers with 64 filters of size 3×3.

Layer	CNN-6	CNN-9	CNN-15
Conv1.x	[3×3, 64]×2	[3×3, 64]×3	[3×3, 64]×5
Pool1	2×2 Max Pooling, Stride 2		
Conv2.x	[3×3, 128]×2	[3×3, 128]×3	[3×3, 128]×5
Pool2	2×2 Max Pooling, Stride 2		
Conv3.x	[3×3, 256]×2	[3×3, 256]×3	[3×3, 256]×5
Pool3	2×2 Max Pooling, Stride 2		
Fully Connected	256	256	256

Table E.2: Our ResNet architectures with different convolutional layers. Conv0.x, Conv1.x, Conv2.x, Conv3.x and Conv4.x denote convolution units that may contain multiple convolutional layers, and residual units are shown in double-column brackets. Conv1.x, Conv2.x and Conv3.x usually operate on different size feature maps. These networks are essentially the same as [2], but some may have a different number of filters in each layer. The downsampling is performed by convolutions with a stride of 2. E.g., [3×3, 64]×4 denotes 4 cascaded convolution layers with 64 filters of size 3×3, S2 denotes stride 2.

Layer	ResNet-32 for CIFAR-10/100	ResNet-18 for ImageNet-2012	ResNet-34 for ImageNet-2012
Conv0.x	N/A	[7×7, 64], Stride 2 3×3, Max Pooling, Stride 2	[7×7, 64], Stride 2 3×3, Max Pooling, Stride 2
Conv1.x	[3×3, 64]×1 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$
Conv2.x	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$
Conv3.x	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$
Conv4.x	N/A	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$
	Average Pooling		

Image recognition settings. The network architectures used in the paper are elaborated

in Table E.1 and Table E.2. For CIFAR-10 and CIFAR-100, we use batch size 128. We start with learning rate 0.1, divide it when the performance is saturated. For ImageNet-2012, we use batch size 64 and start with learning rate 0.1. The learning rate is divided by 10 when the performance is saturated, and the training is terminated at 500k iterations. For ResNet-50 on ImageNet-2012, we use exactly the same architecture as [2]. Note that, for all the compared methods, we always use the best possible hyperparameters to make sure that the comparison is fair. The baseline has exactly the same architecture and training settings as the one that CoMHE uses. For both half-space MHE and all the variants of CoMHE in hidden layers, we set the weighting hyperparameter as 1 in all experiments. [31] already shows that MHE type of losses are not senstitive to the weighting hyperparameter. We use $1e-5$ for the orthonormal and orthogonal regularizations (the hyperparameter is obtained via cross-validation). If not otherwise specified, standard ℓ_2 weight decay ($1e-4$) is applied to all the neural network including baselines and the networks that use MHE regularization. Note that, all the neuron weights in the neural networks used in the paper are not normalized (unless otherwise specified), but both MHE and CoMHE will normalize the neuron weights while computing the regularization loss. For all experiments, we use $s = 2$ in both MHE and CoMHE. As a result, *CoMHE does not need to modify any component of the original neural networks, and it can simply be viewed as an extra regularization loss that can boost the performance*. All the network architectures are implemented by ourselves, so there might be performance difference between our implementation and the original paper due to some different network and optimization hyperparameters. For example, due to the limitation of computation resources, our batch size for ImageNet-2012 is 64 batch size, which might has some performance loss. However, the network and training settings for the baseline and all the compared regularizations are the same, which ensures the fairness of our experiments. In ImageNet classification, we emphasize that our purpose is to compare the gain brought by different regularizations rather than achieving the state-of-the-art performance. We implement the data augmentation in the ImageNet experiment,

following AlexNet [1] and SphereNet [12], so the accuracy may be lower than using the more complicated data augmentation used in original ResNet [2].

Point cloud recognition settings. For all the PointNet and PointNet++ experiments, we exactly follow the same setting in the original papers [158, 159] and their official repositories[1] [2]. Specifically, we combine CoMHE regularization to neurons in all the 1×1 convolution layers before the max pooling layer and the multi-layer perceptron classifier after the max pooling layer. All the regularization is added without changing any components in PointNet. For PointNet experiments, we use point number 1024, batch size 32 and Adam optimizer started with learning rate 0.001, the learning rate will decay by 0.7 every 200k iterations, and the training is terminated at 250 epochs. For PointNet++ experiments, since the MRG (multi-resolution grouping) model is not provided in the official repository, we use the SSG (single scale grouping) model as baseline. Specifically, we use point number 1024, batch size 16 and Adam optimizer started with learning rate 0.001, the learning rate will decay by 0.7 every 200k iterations, and the training is terminated at 251 epochs. For all experiments, we use $s = 1$ in both MHE and CoMHE.

We evaluate on PointNet with T-Net and without T-Net in order to demonstrate that CoMHE is not sensitive to architecture modifications. We follow all the default hyper-parameters used in the official released code, and the only difference is that we further combine an additional regularization loss for the neurons in each layer. One can observe that CoMHE consistently performs better than half-space MHE [31].

Besides PointNet, we combine CoMHE to PointNet++ [159] and further show the improvement of generalization introduced by CoMHE is agnostic to the architecture. We evaluate PointNet++ with and without CoMHE on ModelNet-40. Note that, we exactly follow the released code in the official repository where PointNet++ uses the single scale grouping model. Because the original paper [159] uses the multi-resolution grouping model, the baseline performance reported in our paper is not as good as the accuracy reported in the

[1] https://github.com/charlesq34/pointnet
[2] https://github.com/charlesq34/pointnet2

original paper. However, our purpose is to validate the effectiveness of CoMHE, so we only focus on the performance gain. One can observe that CoMHE achieves about 0.5% accuracy gain, while half-space MHE [31] only has about 0.2% accuracy gain.

E.2 Experimental Details and Full Results of Different Hyperspherical Minimization Strategies

E.2.1 General experimental details

Table E.3: Our small plain CNN architectures with different convolutional layers for the illustrative experiment in Figure 6.1. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers. E.g., [3×3, 64]×3 denotes 3 cascaded convolution layers with 64 filters of size 3×3.

Layer	CNN-3	CNN-9
Conv1.x	[3×3, 32]×1	[3×3, 32]×3
Pool1	2×2 Max Pooling, Stride 2	
Conv2.x	[3×3, 64]×1	[3×3, 64]×3
Pool2	2×2 Max Pooling, Stride 2	
Conv3.x	[3×3, 64]×1	[3×3, 64]×3
Pool3	2×2 Max Pooling, Stride 2	
Fully Connected	64	64

The training details are the same as the CIFAR-100 experiment described in section E.1, except that we use different network structure here. All the optimizations of CNNs use the Stochastic gradient descent with momentum 0.9. The number of iteration and learning rate decay exactly follows the CIFAR-100 experiment in subsubsection 6.6.1. The results in Figure 6.1 are obtained with the CNN-9 described in Table E.3. In order to show that the conclusion obtained using CNN-9 is architecture-agnostic, we also conduct the same experiment on CNN-3. The width of CNN-3 and CNN-9 is smaller than the architectures we used in subsubsection 6.6.1, because the orthogonal training consumes more GPU memory when the size of convolution kernels gets larger. Since the width of CNNs is still the same for all the compared regularizations, it will not affect the validity of our conclusions. For standard, MHE and CoMHE training, the weight decay will be used by default. For rotation training, the weight decay is no longer needed since it does not need to learn the weights

for neurons. Note that, all networks (with different regularization) are initialized with the same weights and therefore have the same hyperspherical energy at the beginning.

E.2.2 Details of different training strategies

Standard Training. In standard training, we use the conventional end-to-end training for all the neurons in the CNN via back-propagation. The standard training is the same as the way that baselines are trained in section 6.6.

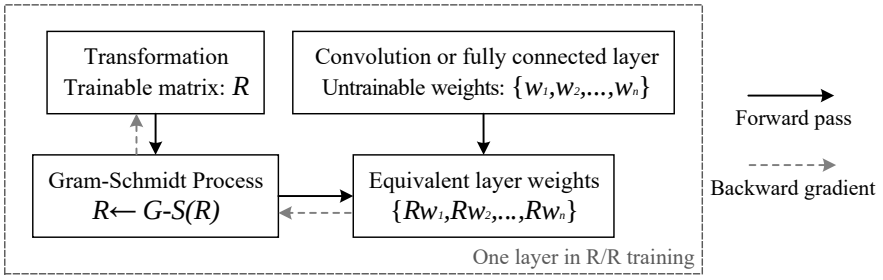


Figure E.1: Illustration of one layer in rotation/reflection training.

Rotation Training. We introduce an interesting baseline here for better comparison to our CoMHE. The core of rotation training [200] is to learn an orthogonal matrix for the neurons in the same layer with the weights of these neurons being fixed. Such an orthogonal matrix is learned individually in every layer except the classifier layer (*i.e.*, the final layer that outputs the class logic). The classifier layer is still learned from scratch via back-propagation. Rotation training is a special case of the orthogonal over-parameterized training in [200]. Specifically, we denote N neurons in the i-th (convolution or fully-connected) layer as $\{\boldsymbol{w}_1^{(i)}, \boldsymbol{w}_2^{(i)}, \cdots, \boldsymbol{w}_N^{(i)}\} \in \mathbb{R}^d$. After randomly initializing these neuron weights with the method in [38], we will fix $\{\boldsymbol{w}_1^{(i)}, \boldsymbol{w}_2^{(i)}, \cdots, \boldsymbol{w}_N^{(i)}\}$ and make them **untrainable** during the entire training procedure. We will learn an orthogonal matrix $\boldsymbol{R}^{(i)} \in \mathbb{R}^{d\times d}$ for the neurons in the i-th layer such that the equivalent neurons become $\{\boldsymbol{R}^{(i)}\boldsymbol{w}_1^{(i)}, \boldsymbol{R}^{(i)}\boldsymbol{w}_2^{(i)}, \cdots, \boldsymbol{R}^{(i)}\boldsymbol{w}_N^{(i)}\}$. The angle between the j-th neuron and k-th neuron is preserved after the rotation/reflection, because we have

$$\cos(\theta_{(\boldsymbol{R}\boldsymbol{w}_j, \boldsymbol{R}\boldsymbol{w}_k)}) = \frac{(\boldsymbol{R}\boldsymbol{w}_j)^\top \boldsymbol{R}\boldsymbol{w}_k}{\|\boldsymbol{R}\boldsymbol{w}_j\| \cdot \|\boldsymbol{R}\boldsymbol{w}_k\|} = \frac{\boldsymbol{w}_j^\top \boldsymbol{w}_k}{\|\boldsymbol{w}_j\| \cdot \|\boldsymbol{w}_k\|} = \cos(\theta_{(\boldsymbol{w}_j, \boldsymbol{w}_k)}). \tag{E.1}$$

Therefore, rotation training only learns the orthogonal matrices $\{\boldsymbol{R}^{(1)}, \boldsymbol{R}^{(2)}, \cdots, \boldsymbol{R}^{(L)}\}$ for neurons of all the convolution and fully-connected layers except the classifier layer. This training strategy will always keep the hyperspherical energy the same during the training process.

For the implementation of rotation training, we use the Gram-Schmidt process to orthonormalizing all the learnable matrices $\boldsymbol{R}^{(i)}, \forall i$ before multiplying them to the neuron weights. Since the Gram-Schmidt process is differentiable, we can directly insert it to process the learnable matrices $\boldsymbol{R}^{(i)}, \forall i$ in the forward pass. We show an overview of forward and backward pass in one layer of rotation training in Figure E.1 to demonstrate the procedure how we orthonormalize the matrices $\boldsymbol{R}^{(i)}, \forall i$ and apply them to the fixed neuron weights.

MHE Training. MHE training is to train the neurons with both data fitting loss and MHE regularization loss from scratch, following the same procedure in [31].

CoMHE Training Similar to MHE training, CoMHE training is to train the neurons with both data fitting loss and CoMHE regularization loss (including RP-CoMHE and AP-CoMHE) from scratch. Details are given in the main paper.

E.2.3 Experimental results

Experiments on CIFAR-100 with CNN-9 (BatchNorm) (also shown in Figure 6.1). We first conduct experiments on CIFAR-100 with CNN-9 as the backbone architecture. For all the compared methods, we use batch normalization. Note that, the experimental results here are the extended results of Figure 6.1. We compute the hyperspherical energy of N neurons using the following definition of half-space hyperspherical energy ($s = 1$) [31] (the same as the regularization loss):

$$\boldsymbol{E} = \frac{1}{2N(2N-1)} \sum_{i=1}^{2N} \sum_{j=1, j\neq i}^{2N} \frac{1}{\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|} \tag{E.2}$$

where $\hat{\boldsymbol{w}}_{N+i} = -\hat{\boldsymbol{w}}_i,\ 0 \leq i \leq N$. This is the hyperspherical energy of neurons in one layer. The total hyperspherical energy needs to sum up the energy from all the layers. We

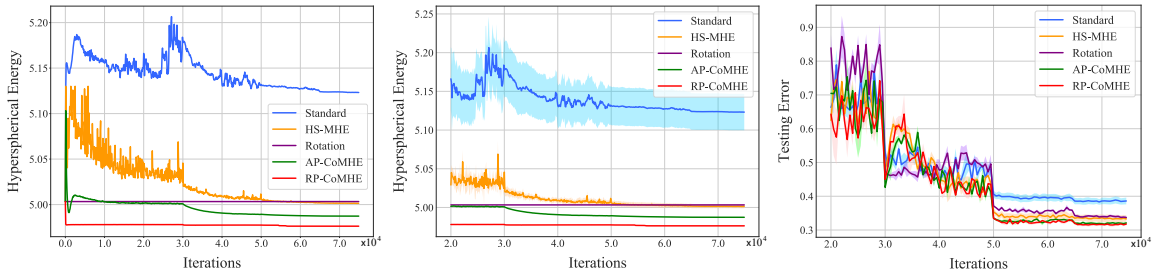


Figure E.2: Results on CIFAR-100 with CNN-9 (BatchNorm). Left: hyperspherical energy v.s. iteration during the entire training. Middle: hyperspherical energy v.s. iteration after the 20000-th iterations (with standard deviation). Right: Testing Error on CIFAR-100 (with standard deviation).

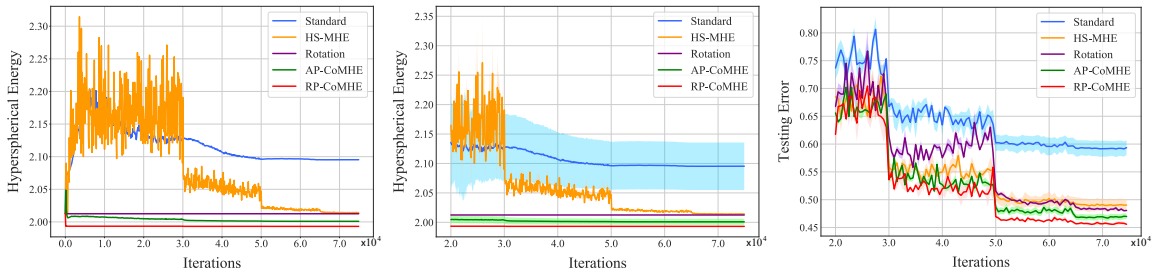


Figure E.3: Results on CIFAR-100 with CNN-3 (BatchNorm). Left: hyperspherical energy v.s. iteration during the entire training. Middle: hyperspherical energy v.s. iteration after the 20000-th iterations (with standard deviation). Right: Testing Error on CIFAR-100 (with standard deviation).

show the hyperspherical energy and the accuracy v.s. iteration in Figure E.2.

The results in Figure E.2 shows that rotation training can largely improve the accuracy compared to the baseline, indicating that the hyperspherical energy can characterize the generalization and lower hyperspherical energy leads to better generalization. The rotation training is able to perform similarly to HS-MHE, showing the advantage of low hyperspherical energy. It also shows that the performance of the original MHE (*i.e.*, HS-MHE) can be achieved by a simple rotation/reflection training strategy.

Experiments on CIFAR-100 with CNN-3 (BatchNorm). To show that the same behavior will also happen in different network structure, we conduct experiments on CIFAR-100 with a 3-layer CNN as shown in Table E.3. Batch normalization is also used. The results in Figure E.3 confirm that rotation training performs better than the baseline and

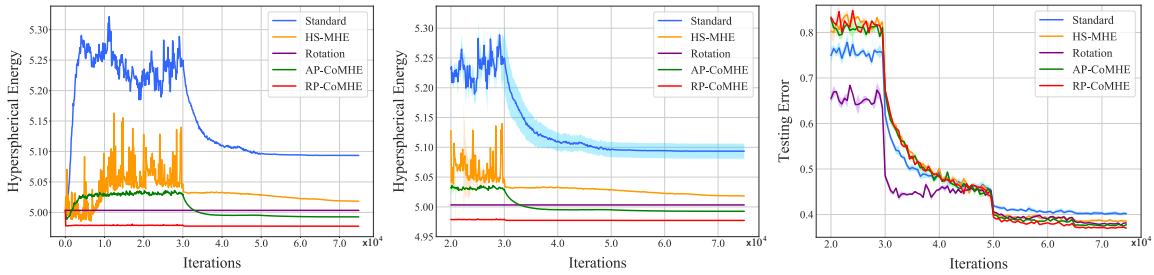


Figure E.4: Results on CIFAR-100 with CNN-9 (no BatchNorm is applied). Left: hyperspherical energy v.s. iteration during the entire training. Middle: hyperspherical energy v.s. iteration after the 20000-th iterations (with standard deviation). Right: Testing Error on CIFAR-100 (with standard deviation).

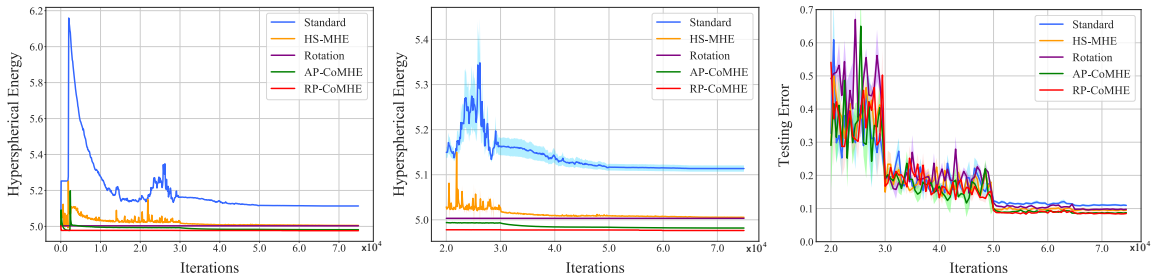


Figure E.5: Results on CIFAR-10 with CNN-9 (BatchNorm). Left: hyperspherical energy v.s. iteration during the entire training. Middle: hyperspherical energy v.s. iteration after the 20000-th iterations (with standard deviation). Right: Testing Error on CIFAR-10 (with standard deviation).

MHE but still worse than our proposed CoMHE. The results verify that hyperspherical energy is an important generaliability indicator for trained networks.

Experiments on CIFAR-100 with CNN-9 (no BatchNorm). To show that batch normalization does not affect our conclusion, we also conduct experiments on CIFAR-100 without using batch normalization. We use the CNN-9 architecture as shown in Table E.3. The results in Figure E.4 show that our conclusion holds even without batch normalization. Hyperspherical energy plays an important role in the network generalization. In general, lower hyperspherical energy leads to better generalization, even though such low hyperspherical energy (in rotation training) is achieved by zero-mean Gaussian initialization.

Experiments on CIFAR-10 with CNN-9 (BatchNorm). To show that the same empirical behavior is consistent in different dataset, we further conduct experiments on CIFAR-

10. We use CNN-9 with batch normalization as our backbone architecture. The results in Figure E.5 show that rotation training still performs much better than the baseline and slightly worse than CoMHE. Most interestingly, rotation training can even perform better than the original MHE.

E.2.4 Conclusion and Discussion

We have extensively tested the hyperspherical energy and accuracy of standard, rotation/reflection, MHE, and CoMHE training under multiple circumstances. The empirical evidences consistently show that hyperspherical energy is of great importance and is able to indicate the potential generalizability of a trained network. Even if we use randomly initialized neurons with low hyperspherical energy, we can still have impressive performance (better than MHE) if proper rotations/reflections of these neurons are learned. Note that rotation/reflection will not change the hyperspherical energy. Therefore, how to effectively minimize the hyperspherical energy is of great significance and is also the central focus of CoMHE.

E.3 Two Random Vectors Are Approximately Orthogonal in High Dimensions

We have two random uniform vectors $\frac{\boldsymbol{X}}{\|\boldsymbol{X}\|}$ and $\frac{\boldsymbol{Y}}{\|\boldsymbol{Y}\|}$ where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are normal distributions. Then the inner product of these two independent unit vectors is $\frac{\langle \boldsymbol{X}, \boldsymbol{Y} \rangle}{\|\boldsymbol{X}\|\|\boldsymbol{Y}\|}$. When $n \to +\infty$, according to the law of large numbers, we have that $\frac{\boldsymbol{X}}{\sqrt{n}} \to 1$ almost surely. By the central limit theorem, $\frac{\langle \boldsymbol{X}, \boldsymbol{Y} \rangle}{\sqrt{n}}$ converges in distribution to a standard one-dimensional normal distribution. Therefore, we have in distribution that

$$\sqrt{n} \cdot \langle \boldsymbol{U}, \boldsymbol{V} \rangle \to z \tag{E.3}$$

where $\boldsymbol{U} = \frac{\boldsymbol{X}}{\|\boldsymbol{X}\|}$, $\boldsymbol{V} = \frac{\boldsymbol{Y}}{\|\boldsymbol{Y}\|}$ and z follows a normal distribution. Then for every $\epsilon > 0$, we have that

$$P(|\langle \boldsymbol{U}, \boldsymbol{V} \rangle|) \to 0 \tag{E.4}$$

which implies that the probability that $\boldsymbol{U}$ and $\boldsymbol{V}$ are nearly orthogonal approaches to 1 when $n \to +\infty$. Similarly, we can also conclude that k independent uniform unit vectors on the hypersphere are nearly orthogonal with very high probability when the dimension becomes higher.

E.4 Johnson-Lindenstrauss Lemma

Lemma 7 (Johnson-Lindenstrauss Lemma [145, 146])**.** *Let $w_1, w_2 \in \mathbb{R}^d$ be vectors, and $\boldsymbol{P} \in \mathcal{R}^{k \times d}, k < d$ be a random projection matrix with entries i.i.d. drawn from a 0-mean σ-subgaussian distribution. With $\boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2 \in \mathbb{R}^k$ being the projected vectors of $\boldsymbol{w}_1, \boldsymbol{w}_2$, then, $\forall \epsilon \in (0, 1)$,*

$$(1 - \epsilon) \left\|\boldsymbol{w}_1 - \boldsymbol{w}_2\right\|^2 k\sigma^2 < \left\|\boldsymbol{P}\boldsymbol{w}_1 - \boldsymbol{P}\boldsymbol{w}_2\right\|^2 < (1 + \epsilon) \left\|\boldsymbol{w}_1 - \boldsymbol{w}_2\right\|^2 k\sigma^2 \tag{E.5}$$

holds with probability at least $1 - 2\exp(-\frac{k\epsilon^2}{8})$.

E.5 Proofs of lemmas and theorems

In the section, we aim to provide the complete proof for self-containedness.

E.5.1 Proof of Lemma 3

We take the expectation of the inner product between projected vectors:

$$\begin{aligned}
&\mathbb{E}(\langle \boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2\rangle) \\
=&\frac{1}{n}\mathbb{E}\Big(\sum_{l=1}^{n}(\sum_{j=1}^{d} r_{lj}\{\boldsymbol{w}_1\}_j \sum_{i=1}^{d} r_{li}\{\boldsymbol{w}_2\}_i)\Big) \\
=&\frac{1}{n}\sum_{l=1}^{n}\Big(\sum_{j=1}^{d}\mathbb{E}(r_{lj}^2)\{\boldsymbol{w}_1\}_j\{\boldsymbol{w}_2\}_j + \sum_{j=1}^{d}\mathbb{E}(r_{lj})\{\boldsymbol{w}_1\}_j \cdot \sum_{i\neq j: i=1}^{d}\mathbb{E}(r_{li})\{\boldsymbol{w}_2\}_i\Big) \\
=&\langle \boldsymbol{w}_1, \boldsymbol{w}_2\rangle
\end{aligned} \tag{E.6}$$

where $\{\boldsymbol{w}_1\}_i$ is the i-th element of the vector $\boldsymbol{w}_1$, and $\{\boldsymbol{w}_2\}_i$ is the i-th element of the vector $\boldsymbol{w}_2$. From the equation, we see that the lemma is proved. □

E.5.2 Proof of Theorem 3

Before proving the main theorem, we first show a lemma from [146].

Lemma 8 (Dot Product under Random Projection)**.** *Let $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^d$, $\boldsymbol{P} \in \mathbb{R}^{k\times d}, k < d$ be a random projection matrix having i.i.d. 0-mean subgaussian entries with parameter σ^2, and $\boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2$ be the images of $\boldsymbol{w}_1, \boldsymbol{w}_2$ under projection $\boldsymbol{P}$. Then, $\forall\epsilon \in (0,1)$:*

$$\boldsymbol{w}_1^\top\boldsymbol{w}_2 k\sigma^2 - \epsilon k\sigma^2 \|\boldsymbol{w}_1\| \|\boldsymbol{w}_2\| < (\boldsymbol{P}\boldsymbol{w}_1)^\top \boldsymbol{P}\boldsymbol{w}_2 < \boldsymbol{w}_1^\top\boldsymbol{w}_2 k\sigma^2 + \epsilon k\sigma^2 \|\boldsymbol{w}_1\| \|\boldsymbol{w}_2\| \tag{E.7}$$

holds with probability $1 - 2\exp(-\frac{k\sigma^2}{8})$.

From Lemma 7, we have that

$$\begin{aligned}
(1-\epsilon)\|\boldsymbol{w}_1\|^2 k\sigma^2 < \|\boldsymbol{P}\boldsymbol{w}_1\|^2 < (1+\epsilon)\|\boldsymbol{w}_1\|^2 k\sigma^2 \\
(1-\epsilon)\|\boldsymbol{w}_2\|^2 k\sigma^2 < \|\boldsymbol{P}\boldsymbol{w}_2\|^2 < (1+\epsilon)\|\boldsymbol{w}_2\|^2 k\sigma^2
\end{aligned} \tag{E.8}$$

which holds with probability $\big(1 - 2\exp(-\frac{k\epsilon^2}{8})\big)^2$.

Then we combine Equation E.8 to Lemma Equation 8 and obtain that

$$\frac{\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)})-\epsilon}{1+\epsilon} < \cos(\theta_{(\boldsymbol{P}\boldsymbol{w}_1,\boldsymbol{P}\boldsymbol{w}_2)}) < \frac{\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)})+\epsilon}{1-\epsilon} \tag{E.9}$$

which holds with probability $\big(1-2\exp(-\frac{k\epsilon^2}{8})\big)^2$. $\theta_{(\boldsymbol{P}\boldsymbol{w}_1,\boldsymbol{P}\boldsymbol{w}_2)}$ denotes the angle between $\boldsymbol{P}\boldsymbol{w}_1$ and $\boldsymbol{P}\boldsymbol{w}_2$, and $\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}$ denotes the angle between $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. □

E.5.3 Proof of Theorem 4

Before proving our main theorem, we first show a lemma [147] below:

Lemma 9. *For any $w \in \mathbb{R}^d$, any random Gaussian matrix $\boldsymbol{P} \in \mathbb{R}^{k\times d}$ where $\boldsymbol{P}_{ij} = \frac{1}{\sqrt{n}} r_{ij}$ and $r_{ij}, \forall i,j$ are i.i.d. random variables from $\mathcal{N}(0,1)$, and $\epsilon \in (0,1)$*

$$\Pr\left((1-\epsilon) \le \frac{\|\boldsymbol{P}\boldsymbol{w}\|^2}{\|\boldsymbol{w}\|^2} \le (1+\epsilon)\right) \ge 1-2\exp\big(-\frac{n}{2}(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3})\big) \tag{E.10}$$

Proof of Lemma 9. From Lemma 3, we have that $\mathbb{E}(\|\boldsymbol{P}\boldsymbol{w}\|^2) = \|\boldsymbol{w}\|^2$. Due to 2-stability of the Gaussian distribution, we have that $\sum_{j=1}^{d} r_{lj} w_j = \|\boldsymbol{w}\| z_l$ where $z_l \sim \mathcal{N}(0,1)$. As a result, we have that

$$\|\boldsymbol{P}\boldsymbol{w}\|^2 = \frac{1}{n}\boldsymbol{w}^2 \sum_{l=1}^{n} z_l^2 \tag{E.11}$$

where $\sum_{l=1}^{n} z_l^2$ is chi-square distributed with n-degree freedom. Then we apply the standard tail bound of the chi-square distribution and obtain

$$\begin{aligned}\Pr\Big(\|\boldsymbol{P}\boldsymbol{w}\|^2 \le (1-\epsilon)\left\|\boldsymbol{w}^2\right\|\Big) &\le \exp\left(\frac{n}{2}\big(1-(1-\epsilon)+\ln(1-\epsilon)\big)\right) \\ &\le \exp(-\frac{n}{4}\epsilon^2)\end{aligned} \tag{E.12}$$

where the inequality $\ln(1-\epsilon) \leq -\epsilon - \frac{\epsilon^2}{2}$ is applied. Similarly, one can have

$$\begin{aligned} \Pr\Big(\|\boldsymbol{P}\boldsymbol{w}\|^2 \leq (1+\epsilon) \left\|\boldsymbol{w}^2\right\| \Big) &\leq \exp\left(\frac{n}{2}\big(1-(1+\epsilon)+\ln(1+\epsilon)\big)\right) \\ &\leq \exp(-\frac{n}{2}(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3})) \end{aligned} \tag{E.13}$$

where the inequality $\ln(1+\epsilon) \leq \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3}$ is used. □

From the lemma above, we apply the union bound and have that

$$\begin{aligned} (1-\epsilon) \leq \frac{\|\boldsymbol{P}\boldsymbol{w}_1\|^2}{\|\boldsymbol{w}_1\|^2} \leq (1+\epsilon) \\ (1-\epsilon) \leq \frac{\|\boldsymbol{P}\boldsymbol{w}_2\|^2}{\|\boldsymbol{w}_2\|^2} \leq (1+\epsilon) \end{aligned} \tag{E.14}$$

which holds with probability at least $1-4\exp(-\frac{n}{2}(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3}))$. Using Equation E.14, we can have that

$$\left\| \frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{P}\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{P}\boldsymbol{w}_2\|} \right\|^2 \leq \left\| \frac{\boldsymbol{P}\boldsymbol{w}_1}{\sqrt{1-\epsilon}\,\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\sqrt{1-\epsilon}\,\|\boldsymbol{w}_2\|} \right\|^2 \tag{E.15}$$

From Equation E.14 and the condition that $\boldsymbol{w}_1^\top \boldsymbol{w}_2 > 0$, we further have that

$$\begin{aligned} \left\| \frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|} \right\|^2 &\leq \left\| \sqrt{1+\epsilon} - \sqrt{1-\epsilon} \right\|^2 \\ &\leq \left\| \sqrt{1+\epsilon} \left(\frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{P}\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{P}\boldsymbol{w}_2\|} \right) \right\|^2 + \left\| \sqrt{1+\epsilon} - \sqrt{1-\epsilon} \right\|^2 \end{aligned} \tag{E.16}$$

Then we apply Lemma 9 to the vector $(\frac{\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|})$ and see that

$$(1-\epsilon) \left\| \frac{\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|} \right\|^2 \leq \left\| \frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|} \right\|^2 \leq (1+\epsilon) \left\| \frac{\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|} \right\|^2 \tag{E.17}$$

which holds with probability $1 - 2\exp\big(-\frac{n}{2}(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3})\big)$. Then we have that

$$\begin{aligned}\frac{\langle \boldsymbol{w}_1, \boldsymbol{w}_2\rangle}{\|\boldsymbol{w}_1\|\,\|\boldsymbol{w}_2\|} &= 1 - \frac{1}{2}\left\|\frac{\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|}\right\|^2, \\ \frac{\langle \boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2\rangle}{\|\boldsymbol{P}\boldsymbol{w}_1\|\,\|\boldsymbol{P}\boldsymbol{w}_2\|} &= 1 - \frac{1}{2}\left\|\frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{P}\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{P}\boldsymbol{w}_2\|}\right\|^2.\end{aligned} \tag{E.18}$$

From Equation E.15, Equation E.16 and Equation E.17, we can learn that $\left\|\frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{P}\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{P}\boldsymbol{w}_2\|}\right\|^2$ is bounded below and above. Further combining Equation E.18, we have that

$$\frac{1+\epsilon}{1-\epsilon}\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}) - \frac{2\epsilon}{1-\epsilon} < \cos(\theta_{(\boldsymbol{P}\boldsymbol{w}_1,\boldsymbol{P}\boldsymbol{w}_2)}) < \frac{1-\epsilon}{1+\epsilon}\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}) + \frac{1+2\epsilon}{1+\epsilon} - \frac{\sqrt{(1-\epsilon^2)}}{1+\epsilon} \tag{E.19}$$

where $\theta_{(\boldsymbol{P}\boldsymbol{w}_1,\boldsymbol{P}\boldsymbol{w}_2)}$ denotes the angle between $\boldsymbol{P}\boldsymbol{w}_1$ and $\boldsymbol{P}\boldsymbol{w}_2$, and $\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}$ denotes the angle between $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. □

E.5.4 Proof of Theorem 5

We first introduce a lemma before proving the theorem.

Lemma 10. *[Direct Result from [201]] Let $\mathcal{H}$ be a separable Hilbert space, and let μ be a non-degenerate Gaussian measure on $\mathcal{H}$. Let P, Q be Borel probability measures on $\mathcal{H}$. Assume that:*

- *The abosolute moments $m_n := \int \|x\|^n \, dP(x)$ are finite and satisfy $\sum_{n\geq 1} m_n^{\frac{-1}{n}} = \infty$;*
- *The set $\varepsilon(P, Q) := \{x \in \mathcal{H} : P_{\langle x\rangle} = Q_{\langle x\rangle}\}$, where $\langle x\rangle$ denotes the one-dimensional subspace spanned by x, is of positive μ-measure.*

Then we have $P = Q$.

If we consider $\boldsymbol{w} \in \mathbb{R}^d$ as a bounded variable, and without loss of generality, we assume that $\boldsymbol{p} = \boldsymbol{z}/\|\boldsymbol{z}\|$ where $\boldsymbol{z}$ is a Gaussian distribution, and then the condition on the moments of $\boldsymbol{w}$ in Lemma 10 holds. Then with the following lemma, we can easily have the desired result.

□

E.6 Bilateral Projection for CoMHE

In this section, we consider bilateral projection for CoMHE (BP-CoMHE) as an extension to the main paper. If we view the neurons in one layer as a matrix $\boldsymbol{W}=\{\boldsymbol{w}_1,\cdots,\boldsymbol{w}_n\}\in\mathbb{R}^{m\times n}$ where m is the dimension of neurons and n is the number of neurons, then the projection mainly considered throughout the paper is to left-multiply a projection matrix $\boldsymbol{P}_1\in\mathbb{R}^{r\times m}$ to $\boldsymbol{W}$. In fact, we can further reduce the number of neurons by right-multiplying an additional projection matrix $\boldsymbol{P}_2\in\mathbb{R}^{n\times r}$ to $\boldsymbol{W}$. Specifically, we denote that

$$\boldsymbol{Y}_1=\boldsymbol{P}_1\boldsymbol{W}\in\mathbb{R}^{r\times n},\quad \boldsymbol{Y}_2=\boldsymbol{W}\boldsymbol{P}_2\in\mathbb{R}^{m\times r} \tag{E.20}$$

BP-CoMHE. The first variant of BP-CoMHE is to apply the MHE regularization separately to column vectors of $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$, and the learning objective is given by

$$\min_{\boldsymbol{W}} \boldsymbol{E}_s(\hat{\boldsymbol{y}}_i^{(1)}|_{i=1}^n)+\boldsymbol{E}_s(\hat{\boldsymbol{y}}_i^{(2)}|_{i=1}^r) \tag{E.21}$$

where we denote that

$$\begin{aligned}
&\boldsymbol{Y}_1=\{\boldsymbol{y}_1^{(1)},\cdots,\boldsymbol{y}_n^{(1)}\}\in\mathbb{R}^{r\times n},\\
&\boldsymbol{Y}_2=\{\boldsymbol{y}_1^{(2)},\cdots,\boldsymbol{y}_r^{(2)}\}\in\mathbb{R}^{m\times r},\\
&\hat{\boldsymbol{Y}}_1=\{\hat{\boldsymbol{y}}_1^{(1)}=\frac{\boldsymbol{y}_1^{(1)}}{\|\boldsymbol{y}_1^{(1)}\|},\cdots,\hat{\boldsymbol{y}}_n^{(1)}=\frac{\boldsymbol{y}_n^{(1)}}{\|\boldsymbol{y}_n^{(1)}\|}\}\in\mathbb{R}^{r\times n},\\
&\hat{\boldsymbol{Y}}_2=\{\hat{\boldsymbol{y}}_1^{(2)}=\frac{\boldsymbol{y}_1^{(2)}}{\|\boldsymbol{y}_1^{(2)}\|},\cdots,\hat{\boldsymbol{y}}_r^{(2)}=\frac{\boldsymbol{y}_r^{(2)}}{\|\boldsymbol{y}_r^{(2)}\|}\}\in\mathbb{R}^{m\times r}.
\end{aligned} \tag{E.22}$$

in which we have that $\hat{\boldsymbol{y}}_i^{(1)}\in\mathbb{R}^{r\times 1}$ and $\hat{\boldsymbol{y}}_i^{(1)}\in\mathbb{R}^{m\times 1}$ are two column vectors of $\hat{\boldsymbol{Y}}_1$ and $\hat{\boldsymbol{Y}}_2$, respectively. The final neurons obtained for the neural network are still $\boldsymbol{W}$. For generating the projection matrices $\boldsymbol{P}_1,\boldsymbol{P}_2$, we simply use random projection and re-initialize the

random matrices every certain number of iterations.

Low-Rank BP-CoMHE. More interestingly, we can also approximate $\boldsymbol{W}$ with a low-rank factorization [154] given as follows:

$$\tilde{\boldsymbol{W}} = \boldsymbol{Y}_2(\boldsymbol{P}_1\boldsymbol{Y}_2)^{-1}\boldsymbol{Y}_1 \in \mathbb{R}^{m\times n} \tag{E.23}$$

which inspires us to directly use two set of parameters $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ to represent the equivalent neurons $\tilde{\boldsymbol{W}}$ and apply the MHE regularization separately to their column vectors (similar to the previous BP-CoMHE). Essentially, we learn the matrices $\boldsymbol{Y}_1, \boldsymbol{Y}_2$ directly via back-propagation. The projection matrix $\boldsymbol{P}_1$ is initialized as a random matrix and stays constant during the training. Different from the former case, we will not use $\boldsymbol{W}$ as the final neurons in the neural network. Instead, we will use $\tilde{\boldsymbol{W}}$ as the final neurons. The number of learnable parameters in total is $mr + nr$, which is significantly lower than the original BP-CoMHE parameterization (*i.e.*, mn) if we choose r to be much smaller than both m and n.

E.7 More Discussion on the Effectiveness of CoMHE

E.7.1 Full results of Figure 6.2

Figure E.6 shows the entire training dynamics (from initialization to the end of training) of the hyperspherical energy of baseline CNN and CNN regularized by orthogonal regularization, HS-MHE, AP-CoMHE and RP-CoMHE. All the networks use exactly the same initialized weights to ensure the hyperspherical energy is the same at the beginning. One can observe that the hyperspherical energy is actually very low for the initialized weights. This is because the initialized weights follows Gaussian distribution and the hyperspherical energy is computed with normalized weights. The normalized weights (sampled from Gaussian distribution) follows the uniform distribution on the hypersphere (see Theorem 1 in [200]), which can obtain the lowest hyperspherical distribution in expectation. However, when the weights of the neural network start to fit the data and minimize the data approx-

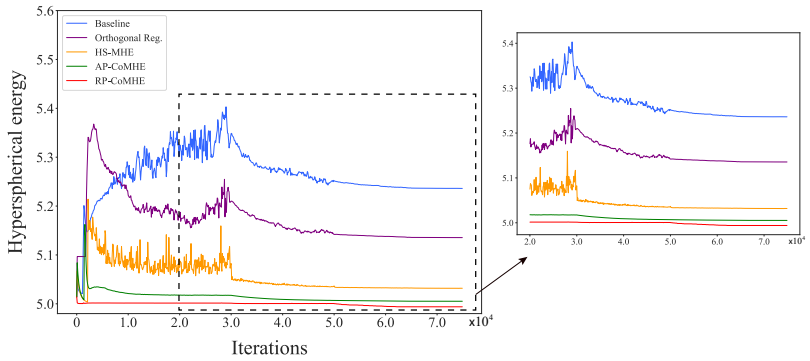


Figure E.6: Hyperspherical energy during the entire training. Note that, all networks are initialized with the same weights and therefore have the same hyperspherical energy at the beginning. Note that, "Orthogonal Reg." denotes the orthogonal regularization (use orthogonal constraint to regularize the neurons), which is dramatically different from the rotation/reflection training that is mentioned above and learns orthogonal matrices for neurons.

imation loss, the neuron weights no longer follow the hyperspherical uniform distribution. Therefore the hyperspherical energy will quickly get large. This is when MHE and CoMHE are useful. From Figure E.6, one can see that without any regularization on hyperspherical energy, the hyperspherical energy of the baseline network gets extremely large at the beginning and then slowly decreases as the training continues. However, the final hyperspherical energy of the baseline network is still way higher than the CNNs regularized by MHE and CoMHE. Notice that, the orthogonality-regularized CNN also obtain high hyperspherical energy at the end (similar to the baseline network). In contrast to MHE, we can observe that CoMHE can effectively minimize the hyperspherical energy and RP-CoMHE achieves significantly lower hyperspherical energy in the end, which well verifies the superiority of the proposed CoMHE.

E.7.2 Hyperspherical energy dynamics in individual layers

To demonstrate the hyperspherical energy dynamics in individual layers, we show the hyperspherical energy v.s. iteration in every layer of CNN-9 (as specified in Table E.1) in Figure E.7. Since the last fully-connected layer (*i.e.*, classifier layer) is learned from scratch (no rotation training is applied), we do not plot its hyperspherical energy. From

the results in Figure E.7, we can observe that CoMHE can more effectively minimize the hyperspherical energy in every layer, and RP-CoMHE performs the best in terms of the hyperspherical energy minimization.

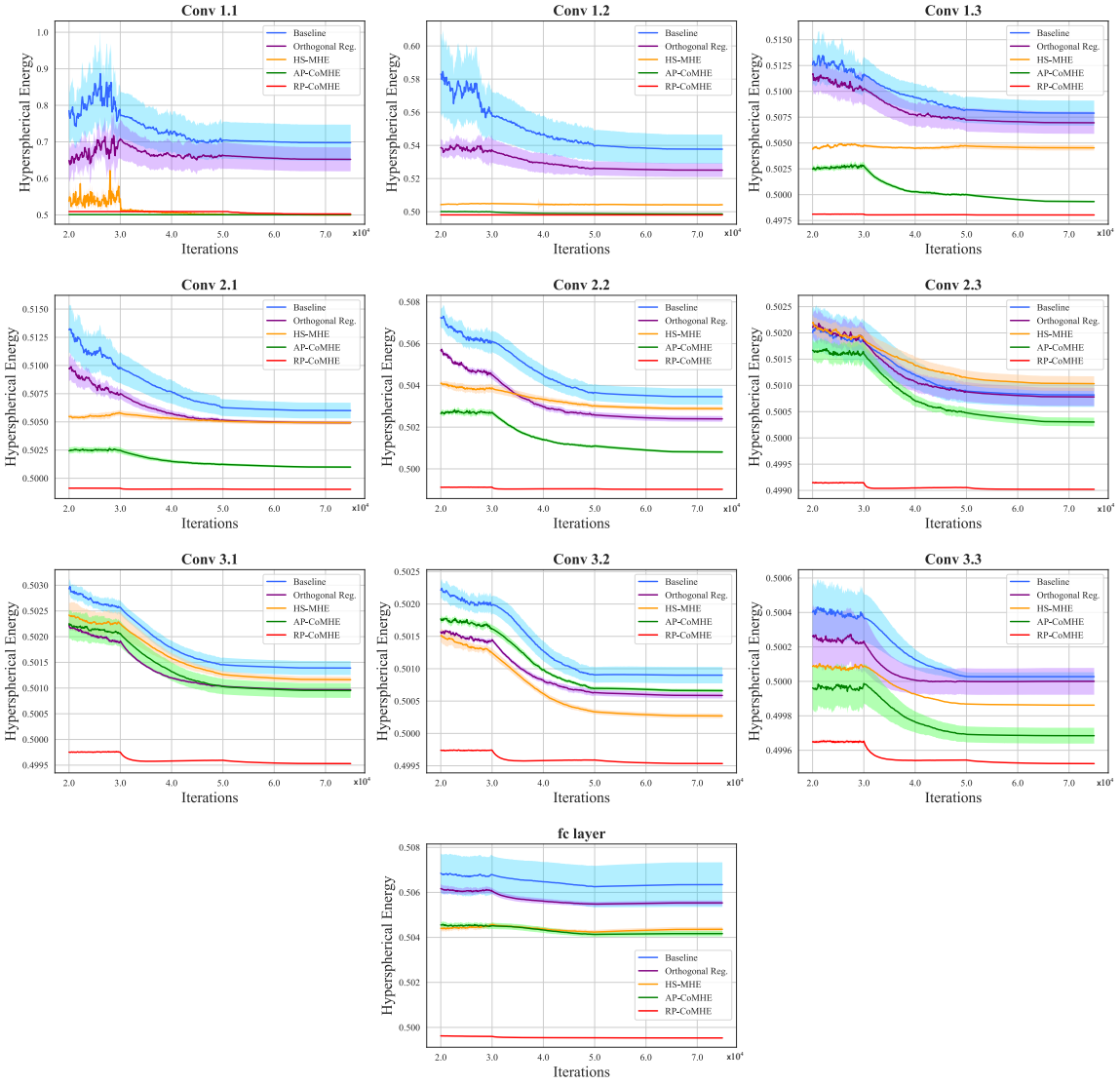


Figure E.7: Hyperspherical energy of every layer (Conv1.1, Conv1.2, Conv1.3, Conv2.1, Conv2.2, Conv2.3, Conv3.1, Conv3.2, Conv3.3, fc1) after the 20000-th iteration. Note that, all networks are initialized with the same weights and therefore have the same hyperspherical energy at the beginning.

E.8 Additional Exploratory Experiments

Frequency of re-initialization in RP-CoMHE. In RP-CoMHE, we need to re-initialize the random projections every certain number of iterations to avoid trivial solutions caused by bad initialization. Here, we test how the frequency of re-initialization will affect the accuracy on CIFAR-100, with the projection dimension being 30 and the number of projection being 20. The iteration number being ∞ in Table E.4 represents that the random projection is fixed throughout the training once it is initialized. The results shows the performance is not very sensitive to the frequency of re-initialization, but we cannot fix the random projection during training as it may cause trivial solutions and hurt the performance.

Table E.4: Error of different # iterations for re-initialization.

# Iterations	1	200	1000	∞
RP-CoMHE	**24.6**	24.84	24.62	26.09

Naively learning projection basis from training data. We study the case where we enable the back-propagation gradient to flow back to the projection basis. That is to say, the model learns the projection basis naively using training data. We find that naively learning the projection basis yields much worse performance (26.5%), compared to RP-CoMHE (24.6%). It is even worse than our baseline half-space MHE (25.96%). The results show that naively learning projection basis from training data leads to inferior performance. Allowing the projection basis to be updated according to the training data could undermine the strength of CoMHE regularization imposed on the neurons.

Shared projection basis. We take RP-CoMHE as an example to empirically verify the advantages of shared projection basis across different layers. We set the projection dimension to 20 and the number of projections to 30. The plain CNN-9 is used as baseline. Specifically for shared projection basis, we share the random projection basis in Conv1.x, Conv2.x and Conv3.x separately. The shared projection yields 24.6% error rate. For in-

dependent projection basis, we use separated projection basis for different layers and only obtain 26.05% error rate. The results show that using shared random projection basis for neurons of the same dimensionality improves the network generalization while saving parameters. Note that, all the other experiments use shared projection basis by default.

E.9 Training Runtime Comparison

We also provide runtime comparison for all the proposed CoMHE. We use the plain CNN-9 for all the methods in this experiment. For RP, we set the projection dimension to 30 and the number of projection to 5. For AP, the number of projection is 1 and the projection dimension is set to 30. This hyperparameter setting for CoMHE can achieve the best testing accuracy on CIFAR-100. The results in Table E.5 are computed using the total runtime of runing 100 iterations. We can see that the runtime of RP-CoMHE, AP-CoMHE and Adv-CoMHE is comparable to HS-MHE and the baseline. Without any code optimization, RP-CoMHE is 36% slower than the baseline and 18% slower than the HS-MHE, and AP-CoMHE is 34% slower than the baseline and 17% slower than the HS-MHE. Note that, although CoMHE is relatively slower in terms of training runtime, CoMHE will not affect the testing runtime of a trained model. That is to say, CoMHE-regularized CNN has the same inference speed with its baseline CNN counterpart. In fact, as long as the training time of CoMHE-regularized CNNs is not geometrically larger than the standard CNN, such computational cost is neglectable in practice and practitioners usually care more about the inference time rather than the training time (CoMHE will not affect the inference time).

Table E.5: Training Runtime (s / 100 iterations) comparison on CIFAR-100.

Method	Runtime (s)
Baseline	5.61
HS-MHE	6.46
RP-CoMHE	7.62
AP-CoMHE	7.48
Adv-CoMHE	6.37
Group CoMHE	11.12

E.10 Experiments on Graph Convolutional Networks

We also use CoMHE to improve graph convolutonal networks (GCN) [161] for node classification in a graph. We use the official code from [161][3], so the experimental setting and hyperparameter setup are exactly the same as [161]. The only difference is that we apply an additional MHE or CoMHE to the weight matrix. Specifically, the graph convolution network uses the following forward model:

$$\boldsymbol{Z} = \text{Softmax}\big(\hat{\boldsymbol{A}} \cdot \text{ReLU}(\hat{\boldsymbol{A}} \cdot \boldsymbol{X} \cdot \boldsymbol{W}_0) \cdot \boldsymbol{W}_1\big) \tag{E.24}$$

where $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{\frac{1}{2}}$. We note that $\boldsymbol{A}$ is the adjacency matrix of the graph, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ ($\boldsymbol{I}$ is an identity matrix), and $\tilde{\boldsymbol{D}} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ is the feature matrix of n nodes in the graph (feature dimension is d). $\boldsymbol{W}_1$ is the weights of the classifiers. $\boldsymbol{W}_0$ is the weight matrix of size $d \times h$ where h is the dimension of the hidden space. We view every column of $\boldsymbol{W}_0$ as a neuron, and therefore, there will be h neurons in total. We simply apply MHE or CoMHE to regularize these h neurons. The experimental results are given in Table E.6. We can see from the results that the CoMHE-regularized GCN can consistently outperform the MHE-regularized GCN and the GCN baseline. We use exactly the same code as in the official repository, and the only difference is the regularization on $\boldsymbol{W}_0$. We emphasize that CoMHE will not change the inference speed of GCN, so this $1\% - 2\%$ performance gain is more like a "free lunch".

Table E.6: Classification accuracy (%) of GCN with different hyperspherical energy regularization.

Method	Citeseer	Cora	Pubmed
GCN Baseline	70.3	81.3	79.0
HS-MHE [31]	71.5	82.0	79.0
RP-CoMHE	**72.1**	**82.7**	**79.5**
AP-CoMHE	72.0	82.6	**79.5**

[3]The code is available at https://github.com/tkipf/gcn.

APPENDIX F
ADDITIONAL RESULTS AND THEORETICAL JUSTIFICATIONS IN CHAPTER 7

F.1 Details of Unrolled Orthogoanlization Algorithms

F.1.1 Gram-Schmidt Process

Gram-Schmidt Process. GS process is a method for orthonormalizing a set of vectors in an inner product space, *i.e.*, the Euclidean space $\mathbb{R}^n$ equipped with the standard inner product. Specifically, GS process performs the following operations to orthogonalize a set of vectors $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\} \in \mathbb{R}^{n\times n}$:

$$
\begin{aligned}
&\text{Step 1: } \tilde{\boldsymbol{e}}_1 = \boldsymbol{u}_1, \quad \boldsymbol{e}_1 = \frac{\tilde{\boldsymbol{e}}_1}{\|\tilde{\boldsymbol{e}}_1\|} \\
&\text{Step 2: } \tilde{\boldsymbol{e}}_2 = \boldsymbol{u}_2 - \text{Proj}_{\tilde{\boldsymbol{e}}_1}(\boldsymbol{u}_2), \quad \boldsymbol{e}_2 = \frac{\tilde{\boldsymbol{e}}_2}{\|\tilde{\boldsymbol{e}}_2\|} \\
&\text{Step 3: } \tilde{\boldsymbol{e}}_3 = \boldsymbol{u}_3 - \text{Proj}_{\tilde{\boldsymbol{e}}_1}(\boldsymbol{u}_3) - \text{Proj}_{\tilde{\boldsymbol{e}}_2}(\boldsymbol{u}_3), \quad \boldsymbol{e}_3 = \frac{\tilde{\boldsymbol{e}}_3}{\|\tilde{\boldsymbol{e}}_3\|} \\
&\text{Step 4: } \tilde{\boldsymbol{e}}_4 = \boldsymbol{u}_4 - \text{Proj}_{\tilde{\boldsymbol{e}}_1}(\boldsymbol{u}_4) - \text{Proj}_{\tilde{\boldsymbol{e}}_2}(\boldsymbol{u}_4) - \text{Proj}_{\tilde{\boldsymbol{e}}_3}(\boldsymbol{u}_4), \quad \boldsymbol{e}_4 = \frac{\tilde{\boldsymbol{e}}_4}{\|\tilde{\boldsymbol{e}}_4\|} \\
&\qquad \vdots \\
&\text{Step n: } \tilde{\boldsymbol{e}}_n = \boldsymbol{u}_n - \text{Proj}_{\tilde{\boldsymbol{e}}_1}(\boldsymbol{u}_n) - \text{Proj}_{\tilde{\boldsymbol{e}}_2}(\boldsymbol{u}_n) - \text{Proj}_{\tilde{\boldsymbol{e}}_3}(\boldsymbol{u}_n) - \cdots - \text{Proj}_{\tilde{\boldsymbol{e}}_{n-1}}(\boldsymbol{u}_n), \quad \boldsymbol{e}_n = \frac{\tilde{\boldsymbol{e}}_n}{\|\tilde{\boldsymbol{e}}_n\|}
\end{aligned}
\tag{F.1}
$$

where $\text{Proj}_{\boldsymbol{a}}(\boldsymbol{b}) = \frac{\langle \boldsymbol{a}, \boldsymbol{b}\rangle}{\langle \boldsymbol{a}, \boldsymbol{a}\rangle}\boldsymbol{a}$ denotes the projection of the vector $\boldsymbol{b}$ onto the vector $\boldsymbol{a}$. The set $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_n\}$ denotes the output orthonormal set. The algorithm flowchart can be described as follows:

Algorithm 1 Gram-Schmidt Process

Input: $\boldsymbol{U} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_n\} \in \mathbb{R}^{n \times n}$

Output: $\boldsymbol{R} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_n\} \in \mathbb{R}^{n \times n}$

$\boldsymbol{R} = \boldsymbol{0}$

for $j = 1, 2, \cdots, n$ **do**

$\boldsymbol{q}_j = \boldsymbol{R}^\top \boldsymbol{u}_j$

$\boldsymbol{t} = \boldsymbol{u}_j - \boldsymbol{R}\boldsymbol{q}_j$

$\boldsymbol{q}_{jj} = \|\boldsymbol{t}\|_2$

$\boldsymbol{e}_j = \frac{\boldsymbol{t}}{\boldsymbol{q}_{jj}}$

end

The vectors $\boldsymbol{q}_j, \forall j$ in the algorithm above are used to compute the QR factorization, which is not useful in orthogonalization and therefore does not need to be stored. When the GS process is implemented on a finite-precision computer, the vectors $\boldsymbol{e}_j, \forall j$ are often not quite orthogonal, because of rounding errors. Besides the standard GS process, there is a modified Gram-Schmidt (MGS) algorithm which enjoys better numerical stability. This approach gives the same result as the original formula in exact arithmetic and introduces smaller errors in finite-precision arithmetic. Specifically, GS computes the following formula:

$$\begin{aligned} \tilde{\boldsymbol{e}}_j &= \boldsymbol{u}_j - \sum_{k=1}^{j-1} \text{Proj}_{\tilde{\boldsymbol{e}}_k}(\boldsymbol{u}_j) \\ \boldsymbol{e}_j &= \frac{\tilde{\boldsymbol{e}}_j}{\|\tilde{\boldsymbol{e}}_j\|} \end{aligned} \tag{F.2}$$

Instead of computing the vector $\boldsymbol{e}_j$ as in Equation F.2, MGS computes the orthogonal basis differently. MGS does not subtract the projections of the original vector set, and instead remove the projection of the previously constructed orthogonal basis. Specifically, MGS

computes the following series of formulas:

$$
\begin{aligned}
\tilde{\boldsymbol{e}}_j^{(1)} &= \boldsymbol{u}_j - \text{Proj}_{\tilde{e}_1}(\boldsymbol{u}_j) \\
\tilde{\boldsymbol{e}}_j^{(2)} &= \tilde{\boldsymbol{e}}_j^{(1)} - \text{Proj}_{\tilde{e}_2}(\tilde{\boldsymbol{e}}_j^{(1)}) \\
&\vdots \\
\tilde{\boldsymbol{e}}_j^{(j-2)} &= \tilde{\boldsymbol{e}}_j^{(j-3)} - \text{Proj}_{\tilde{e}_2}(\tilde{\boldsymbol{e}}_j^{(j-3)}) \\
\tilde{\boldsymbol{e}}_j^{(j-1)} &= \tilde{\boldsymbol{e}}_j^{(j-2)} - \text{Proj}_{\tilde{e}_2}(\tilde{\boldsymbol{e}}_j^{(j-2)}) \\
\boldsymbol{e}_j &= \frac{\tilde{\boldsymbol{e}}_j^{(j-1)}}{\left\| \tilde{\boldsymbol{e}}_j^{(j-1)} \right\|}
\end{aligned}
\tag{F.3}
$$

where each step finds a vector $\tilde{\boldsymbol{e}}_j^{(i)}$ that is orthogonal to $\tilde{\boldsymbol{e}}_j^{(i-1)}$. Therefore, $\tilde{\boldsymbol{e}}_j^{(i)}$ is also orthogonalized against any errors brought by the computation of $\tilde{\boldsymbol{e}}_j^{(i-1)}$. In practice, although MGS enjoys better numerical stability, we find the empirical performance of GS and MGS is almost the same in OPT. However, MGS takes longer time to complete since the computation of each orthogonal basis is an iterative process. Therefore, we usually stick to classic GS for OPT.

Iterative Gram-Schmidt Process. Iterative Gram-Schmidt (IGS) process is an iterative version of the GS process. It is shown in [178] that GS process can be carried out iteratively to obtain a basis matrix that is orthogonal in almost full working precision. The IGS algorithm is given as follows:

Algorithm 2 Iterative Gram-Schmidt Process

Input: $\boldsymbol{U} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_n\} \in \mathbb{R}^{n \times n}$

Output: $\boldsymbol{R} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_n\} \in \mathbb{R}^{n \times n}$

$\boldsymbol{R} = \boldsymbol{0}$

for $j = 1, 2, \cdots, n$ **do**

  $\boldsymbol{q}_j = \boldsymbol{0}$

  $\boldsymbol{t} = \boldsymbol{u}_j$

  while $\boldsymbol{t} \perp \text{span}(\boldsymbol{e}_1, \cdots, \boldsymbol{e}_{j-1})$ *is False* **do**

    $\boldsymbol{p} = \boldsymbol{t}$

    $\boldsymbol{s} = \boldsymbol{R}^\top \boldsymbol{p}$

    $\boldsymbol{v} = \boldsymbol{R}\boldsymbol{s}$

    $\boldsymbol{t} = \boldsymbol{p} - \boldsymbol{v}$

    $\boldsymbol{q}_j \leftarrow \boldsymbol{q}_j + \boldsymbol{s}$

  end

  $\boldsymbol{q}_{jj} = \|\boldsymbol{t}\|_2$

  $\boldsymbol{e}_j = \frac{\boldsymbol{t}}{\boldsymbol{q}_{jj}}$

end

The vectors $\boldsymbol{q}_j, \forall j$ in the algorithm above are used to compute the QR factorization, which is not useful in orthogonalization and therefore does not need to be explicitly computed. The while loop in IGS is an iterative procedure. In practice, we can unroll a fixed number of steps for the while loop in order to improve the orthogonality. The resulting $\boldsymbol{q}_j$ in the j-th step corresponds to the solution of the equation $\tilde{\boldsymbol{R}}^\top \tilde{\boldsymbol{R}} \boldsymbol{q}_j = \tilde{\boldsymbol{R}}^\top \boldsymbol{u}_j$ where $\tilde{\boldsymbol{R}} = \{\boldsymbol{e}_1, \cdots, \boldsymbol{e}_{j-1}\}$. The IGS process corresponds to the Gauss-Jacobi iteration for solving this equation.

Both GS and IGS are easy to be embedded in the neural networks, since they are both differentiable. In our experiments, we find that the performance gain of unrolling multiple steps in IGS over GS is not very obvious (partially because GS has already achieved nearly perfect orthogonality), but IGS costs longer training time. Therefore, we unroll the classic

GS process by default.

F.1.2 Householder Reflection

Let $\boldsymbol{v} \in \mathbb{R}^n$ be a non-zero vector. A matrix $\boldsymbol{H} \in \mathbb{R}^{n \times n}$ of the form

$$\boldsymbol{H} = \boldsymbol{I} - \frac{2\boldsymbol{v}\boldsymbol{v}^\top}{\boldsymbol{v}^\top \boldsymbol{v}} \tag{F.4}$$

is a Householder reflection. The vector $\boldsymbol{v}$ is the Householder vector. If a vector $\boldsymbol{x}$ is multiplied by the matrix $\boldsymbol{H}$, then it will be reflected in the hyperplane span$(\boldsymbol{v})^\perp$. Householder matrices are symmetric and orthogonal.

For a vector $\boldsymbol{x} \in \mathbb{R}^n$, we let $\boldsymbol{v} = \boldsymbol{x} \pm \|\boldsymbol{x}\|_2 \boldsymbol{e}_1$ where $\boldsymbol{e}_1$ is a vector of $\{1, 0, \cdots, 0\}$ (the first element is 1 and the remaining elements are 0). Then we construct the Householder reflection matrix with $\boldsymbol{v}$ and multiply it to $\boldsymbol{x}$:

$$\boldsymbol{H}\boldsymbol{x} = \left(\boldsymbol{I} - 2\frac{\boldsymbol{v}\boldsymbol{v}^\top}{\boldsymbol{v}^\top \boldsymbol{v}}\right)\boldsymbol{x} = \mp \|\boldsymbol{x}\|_2 \boldsymbol{e}_1 \tag{F.5}$$

which indicates that we can make any non-zero vector become $\alpha \boldsymbol{e}_1$ where α is some constant by using Householder reflection. By left-multiplying a reflection we can turn a dense vector $\boldsymbol{x}$ into a vector with the same length and with only a single nonzero entry. Repeating this n times gives us the Householder QR factorization, which also orthogonalizes the original input matrix. Householder reflection orthogonalizes a matrix $\boldsymbol{U} = \{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\}$ by triangularizing it:

$$\boldsymbol{U} = \boldsymbol{H}_1 \boldsymbol{H}_2 \cdots \boldsymbol{H}_n \boldsymbol{R} \tag{F.6}$$

where $\boldsymbol{R}$ is a upper-triangular matrix in the QR factorization. $\boldsymbol{H}_j, j \geq 2$ is constructed by Diag$(\boldsymbol{I}_{j-1}, \tilde{\boldsymbol{H}}_{n-j+1})$ where $\tilde{\boldsymbol{H}}_{n-j+1} \in \mathbb{R}^{(n-j+1)\times(n-j+1)}$ is the Householder reflection that is performed on the vector $\boldsymbol{U}_{(j:n,j)}$. The algorithm flowchart is given as follows:

Algorithm 3 Householder Reflection Orthogonalization

Input: $\boldsymbol{U} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_n\} \in \mathbb{R}^{n \times n}$

Output: $\boldsymbol{U} = \boldsymbol{Q}\boldsymbol{R}$, where $\boldsymbol{Q} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_n\} \in \mathbb{R}^{n \times n}$ is the orthogonal matrix and $\boldsymbol{R} \in \mathbb{R}^{n \times n}$ is a upper triangular matrix

for $j = 1, 2, \cdots, n-1$ **do**

$\{\boldsymbol{v}, \beta\} = \texttt{Householder}(\boldsymbol{U}_{j:n,j})$

$\boldsymbol{U}_{j:n,j:n} \leftarrow \boldsymbol{U}_{j:n,j:n} - \beta \boldsymbol{v}(\boldsymbol{v}^\top \boldsymbol{U}_{j:n,j:n})$

$\boldsymbol{U}_{j+1:n,j} \leftarrow \boldsymbol{v}_{(2:\text{end})}$

end

function $\{\boldsymbol{v}, \beta\} = \texttt{Householder}(\boldsymbol{x})$

$\sigma^2 = \|\boldsymbol{x}_{2:\text{end}}\|_2^2$

$\boldsymbol{v} \leftarrow \begin{bmatrix} 1 \\ \boldsymbol{x}_{2:\text{end}} \end{bmatrix}$

if $\sigma^2 = 0$ **then**

$\beta = 0$

else

if $\boldsymbol{x}_1 \leq 0$ **then**

$\boldsymbol{v}_1 = \boldsymbol{x}_1 - \sqrt{\boldsymbol{x}_1^2 + \sigma^2}$

else

$\boldsymbol{v}_1 = -\dfrac{\sigma^2}{\boldsymbol{x}_1 + \sqrt{\boldsymbol{x}_1^2 + \sigma^2}}$

end

$\beta = \frac{2\boldsymbol{v}_1^2}{\sigma^2 + \boldsymbol{v}_1^2}$

$\boldsymbol{v} \leftarrow \frac{\boldsymbol{v}}{\boldsymbol{v}_1}$

end

end function

The algorithm follows the Matlab notation where $\boldsymbol{U}_{j:n,j:n}$ denotes the submatrix of $\boldsymbol{U}$ from the j-th column to the n-th column and from the j-th row to the n-th row. Note that,

there are a number of variants for the Householder reflection orthogonalization, such as the implicit variant where we do not store each reflection $\boldsymbol{H}_j$ explicitly. Here $\boldsymbol{Q}$ is the final orthogonal matrix we need.

F.1.3 Löwdin's Symmetric Orthogonalization

Let $\boldsymbol{U} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_n\}$ be a set of linearly independent vectors in a n-dimensional space. We define a general non-singular linear transformation $\boldsymbol{A}$ that can transform the basis $\boldsymbol{U}$ to a new basis $\boldsymbol{R}$:

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{A} \tag{F.7}$$

where the basis $\boldsymbol{R}$ will be orthonormal if (the transpose will become conjugate transpose in complex space)

$$\boldsymbol{R}^\top \boldsymbol{R} = (\boldsymbol{U}\boldsymbol{A})^\top (\boldsymbol{U}\boldsymbol{A}) = \boldsymbol{A}^\top \boldsymbol{U}^\top \boldsymbol{U}\boldsymbol{A} = \boldsymbol{A}^\top \boldsymbol{M}\boldsymbol{A} = \boldsymbol{I} \tag{F.8}$$

where $\boldsymbol{M} = \boldsymbol{U}^\top \boldsymbol{U}$ is the gram matrix of the given basis $\boldsymbol{U}$.

A general solution to this orthogonalization problem can be obtained via the substitution:

$$\boldsymbol{A} = \boldsymbol{M}^{-1}\boldsymbol{B} \tag{F.9}$$

in which $\boldsymbol{B}$ is an arbitrary orthogonal (or unitary) matrix. When $\boldsymbol{B} = \boldsymbol{I}$, we will have the symmetric orthogonalization, namely

$$\boldsymbol{R} := \boldsymbol{\Phi} = \boldsymbol{U}\boldsymbol{M}^{-\frac{1}{2}} \tag{F.10}$$

When $\boldsymbol{B} = \boldsymbol{V}$ in which $\boldsymbol{V}$ diagonalizes $\boldsymbol{M}$, then we have the canonical orthogonalization, namely

$$\boldsymbol{\Lambda} = \boldsymbol{U}\boldsymbol{V}\boldsymbol{d}^{-\frac{1}{2}}. \tag{F.11}$$

Because $\boldsymbol{V}$ diagonalizes $\boldsymbol{M}$, we have that $\boldsymbol{M} = \boldsymbol{V}\boldsymbol{d}\boldsymbol{V}^\top$. Therefore, we have the $\boldsymbol{M}^{-\frac{1}{2}}$ transformation as $\boldsymbol{M}^{-\frac{1}{2}} = \boldsymbol{V}\boldsymbol{d}^{-\frac{1}{2}}\boldsymbol{V}^\top$. This is essentially an eigenvalue decomposition of the symmetric matrix $\boldsymbol{M} = \boldsymbol{U}^\top\boldsymbol{U}$.

In order to compute the Löwdin's symmetric orthogonalized basis sets, we can use singular value decomposition. Specifically, SVD of the original basis set $\boldsymbol{U}$ is given by

$$\boldsymbol{U} = \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{V}^\top \tag{F.12}$$

where both $\boldsymbol{W} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{U} \in \mathbb{R}^{n\times n}$ are orthogonal matrices. $\boldsymbol{\Sigma}$ is the diagonal matrix of singular values. Therefore, we have that

$$\begin{aligned}\boldsymbol{R} &= \boldsymbol{U}\boldsymbol{M}^{-\frac{1}{2}} \\ &= \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{d}^{-\frac{1}{2}}\boldsymbol{V}^\top \\ &= \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{d}^{-\frac{1}{2}}\boldsymbol{V}^\top\end{aligned} \tag{F.13}$$

where we have $\boldsymbol{\Sigma} = \boldsymbol{d}^{\frac{1}{2}}$ due to the connections between eigenvalue decomposition and SVD. Therefore, we end up with

$$\boldsymbol{R} = \boldsymbol{W}\boldsymbol{V}^\top \tag{F.14}$$

which is the output orthogonal matrix for Löwdin's symmetric orthogonalization.

An interesting feature of the symmetric orthogonalization is to ensure that

$$\boldsymbol{R} = \arg\min_{\boldsymbol{P}\in\text{orth}(\boldsymbol{U})} \sum_i \|\boldsymbol{P}_i - \boldsymbol{U}_i\| \tag{F.15}$$

where $\boldsymbol{P}_i$ and $\boldsymbol{U}_i$ are the i-th column vectors of $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{U}$, respectively. orth$(\boldsymbol{U})$ denotes the set of all possible orthonormal sets in the range of $\boldsymbol{U}$. This means that the symmetric orthogonalization functions $\boldsymbol{R}_i$ (or $\boldsymbol{\Phi}_i$) are the least distant in the Hilbert space from the original functions $\boldsymbol{U}_i$. Therefore, symmetric orthogonalization indicates the gentlest pushing of the directions of the vectors in order to make them orthogonal.

More interestingly, the symmetric orthogonalized basis sets has unique geometric properties [202, 203] if we consider the Schweinler-Wigner matrix in terms of the sum of squared projections.

F.2 Proof of Theorem 6

To be more specific, neurons with each element initialized by a zero-mean Gaussian distribution are uniformly distributed on a hypersphere. We show this argument with the following theorem.

Theorem 7. *The normalized vector of Gaussian variables is uniformly distributed on the sphere. Formally, let* $x_1, x_2, \cdots, x_n \sim \mathcal{N}(0,1)$ *and be independent. Then the vector*

$$\boldsymbol{x} = \left[\frac{x_1}{z}, \frac{x_2}{z}, \cdots, \frac{x_n}{z}\right] \tag{F.16}$$

follows the uniform distribution on $\mathbb{S}^{n-1}$*, where* $z = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$ *is a normalization factor.*

Proof. A random variable has distribution $\mathcal{N}(0,1)$ if it has the density function

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}. \tag{F.17}$$

A n-dimensional random vector $\boldsymbol{x}$ has distribution $\mathcal{N}(0,1)$ if the components are independent and have distribution $\mathcal{N}(0,1)$ each. Then the density of $\boldsymbol{x}$ is given by

$$f(x) = \frac{1}{(\sqrt{2\pi})^n} e^{-\frac{1}{2}\langle x,x\rangle}. \tag{F.18}$$

Then we introduce the following lemma (Lemma 11) about the orthogonal-invariance of the normal distribution.

Lemma 11. *Let* $\boldsymbol{x}$ *be a* n*-dimensional random vector with distribution* $\mathcal{N}(0,1)$ *and* $\boldsymbol{U} \in$

$\mathbb{R}^{n\times n}$ *be an orthogonal matrix (*$\boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{I}$*). Then* $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{x}$ *also has the distribution of* $\mathcal{N}(0,1)$.

Proof. For any measurable set $A \subset \mathbb{R}^n$, we have that

$$\begin{aligned} P(Y \in A) &= P(X \in U^\top A) \\ &= \int_{U^\top A} \frac{1}{(\sqrt{2\pi})^n} e^{-\frac{1}{2}\langle x,x\rangle} \\ &= \int_{A} \frac{1}{(\sqrt{2\pi})^n} e^{-\frac{1}{2}\langle Ux,Ux\rangle} \\ &= \int_{A} \frac{1}{(\sqrt{2\pi})^n} e^{-\frac{1}{2}\langle x,x\rangle} \end{aligned} \tag{F.19}$$

because of orthogonality of U. Therefore the lemma holds. □

Because any rotation is just a multiplication with some orthogonal matrix, we know that normally distributed random vectors are invariant to rotation. As a result, generating $\boldsymbol{x} \in \mathbb{R}^n$ with distribution $\mathbb{N}(0,1)$ and then projecting it onto the hypersphere $\mathbb{S}^{n-1}$ produces random vectors $U = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$ that are uniformly distributed on the hypersphere. Therefore the theorem holds. □

Then we show the normalized vector $\boldsymbol{y}$ where each element follows a zero-mean Gaussian distribution with some constant variance σ^2:

$$\boldsymbol{y} = \left[\frac{y_1}{r}, \frac{y_2}{r}, \cdots, \frac{y_n}{r}\right] \tag{F.20}$$

where $r = \sqrt{y_1^2 + y_2^2 + \cdots + y_n^2}$. Because we have that $\frac{y_i}{\sigma} \sim \mathcal{N}(0,1)$, we can rewrite $\boldsymbol{y}$ as the following random vector:

$$\boldsymbol{y} = \left[\frac{y_1/\sigma}{r/\sigma}, \frac{y_2/\sigma}{r/\sigma}, \cdots, \frac{y_n/\sigma}{r/\sigma}\right] \tag{F.21}$$

where $r/\sigma = \sqrt{(y_1/\sigma)^2 + (y_2/\sigma)^2 + \cdots + (y_n/\sigma)^2}$. Therefore, we directly can apply Theorem Theorem 7 and conclude that $\boldsymbol{y}$ also follows the uniform distribution on $\mathbb{S}^{n-1}$.

Now we obtain that any random vector with each element following a zero-mean Gaussian distribution with some constant variance follows the uniform distribution on $\mathbb{S}^{n-1}$.

Then we show that the minimum hyperspherical energy asymptotically corresponds to the uniform distribution over the unit hypersphere. We first write down the hyperspherical energy of N neurons $\{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_N \in \mathbb{R}^{d+1}\}$ (we also define that $\hat{\boldsymbol{w}}_i = \frac{\boldsymbol{w}_i}{\|\boldsymbol{w}_i\|} \in \mathbb{S}^d$):

$$\boldsymbol{E}_{s,d}(\hat{\boldsymbol{w}}_i|_{i=1}^N) = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} f_s\big(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|\big) = \begin{cases} \sum_{i\neq j} \|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-s}, & s > 0 \\ \sum_{i\neq j} \log\big(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-1}\big), & s = 0 \end{cases} \tag{F.22}$$

where s is a hyperparameter that controls the behavior of hyperspherical energy. We then define a N-point minimal hyperspherical s-energy over $\boldsymbol{A}$ with

$$\boldsymbol{\varepsilon}_{s,d}(\boldsymbol{A}, \hat{\boldsymbol{W}}_N) := \inf_{\hat{\boldsymbol{W}}_N \subset \boldsymbol{A}} \boldsymbol{E}_{s,d}(\hat{\boldsymbol{w}}_i|_{i=1}^N) \tag{F.23}$$

where we denote that $\hat{\boldsymbol{W}}_N = \{\hat{\boldsymbol{w}}_1, \cdots, \hat{\boldsymbol{w}}_N\}$. Typically, we will assume that $\boldsymbol{A}$ is compact. Based on [204], we discuss the asymptotic behavior (as $N \to \infty$) of $\boldsymbol{\varepsilon}_{s,d}(\boldsymbol{A}, \hat{\boldsymbol{W}}_N)$ in three different scenarios: (1) $0 < s < d$; (2) $s = d$; and $s > d$. The reason behind is the behavior of the following energy integral:

$$I_s(\mu) = \iint_{\mathbb{S}^d \times \mathbb{S}^d} \|\boldsymbol{u} - \boldsymbol{v}\|^{-s} d\mu(\boldsymbol{u}) d\mu(\boldsymbol{v}), \tag{F.24}$$

is quite different under these three scenarios. In scenario (1), Equation F.24 that is taken over all probability measures μ supported on $\mathcal{S}^d$ will be minimal for normalized Lebesgue measure $\frac{\mathcal{H}_d(\cdot)|_{\mathbb{S}^d}}{\mathcal{H}_d(\mathbb{S}^d)}$ on $\mathbb{S}^d$. In the case of $s \geq d$, we will have that $I_s(\mu)$ is positive infinity for all such measures μ. Therefore, the behaviour of the minimum hyperspherical energy is different in these three cases. In general, as the parameter s increases, there is a transition from the global effects to the more local influences (from nearest neighbors). The transition happens when $s = d$. However, we typically have $0 < s < d$ in the neural networks.

Therefore, we will mostly study the case of $0 < s < d$ and the theoretical asymptotic behavior is quite standard results from the potential theory [205]. From the classic potential theory, we have the following known lemma:

Lemma 12. *If $0 < s < d$, we have that*

$$\lim_{N\to\infty} \frac{\boldsymbol{\varepsilon}_{s,d}(\boldsymbol{S}^d, \hat{\boldsymbol{W}}_N)}{N^2} = I_s\left(\frac{\mathcal{H}_d(\cdot)|_{\mathbb{S}^d}}{\mathcal{H}_d(\mathbb{S}^d)}\right) \tag{F.25}$$

Moreover, any sequence of s-energy configuration of minimal hyperspherical energy ($(\hat{\boldsymbol{W}}_N^*)_2^\infty \subset \mathbb{S}^d$) is asymptotically uniformly distributed in the sense that for the weak-star topology of measures,

$$\frac{1}{N}\sum_{\boldsymbol{v}\in\hat{\boldsymbol{W}}_N^*} \delta_{\boldsymbol{v}} \to \frac{\mathcal{H}_d(\cdot)|_{\mathbb{S}^d}}{\mathcal{H}_d(\mathbb{S}^d)} \quad \text{as } N \to \infty \tag{F.26}$$

where $\delta_{\boldsymbol{v}}$ denotes the unit point mass at $\boldsymbol{v}$.

The lemma above concludes that the neuron configuration with minimal hyperspherical energy asymptotically corresponds to the uniform distribution on $\mathbb{S}^d$ when $0 < s < d$. From [204], we also have the following lemma that shows the same conclusion holds for the the case of $s = d$ and $s > d$:

Lemma 13. *Let $\mathcal{B}^d := \bar{B}(0, 1)$ denote the closed unit ball in $\mathbb{R}^d$. For the case of $s = d$, we have that*

$$\lim_{N\to\infty} \frac{\boldsymbol{\varepsilon}_{s,d}(\boldsymbol{S}^d, \hat{\boldsymbol{W}}_N)}{N^2 \log N} = \frac{\mathcal{H}_d(\mathcal{B}^d)}{\mathcal{H}_d(\mathbb{S}^d)} = \frac{1}{d}\frac{\Gamma(\frac{d+1}{2})}{\sqrt{\pi}\Gamma(\frac{d}{2})} \tag{F.27}$$

and any sequence $(\hat{\boldsymbol{W}}_N^*) \subset \mathbb{S}^d$ of minimal s-energy configurations satisfies Equation F.26.

The lemma above shows that the same conclusion holds for $s = d$. For the case of $s > d$, the theoretical analysis is more involved, but the conclusion that the neuron configuration with minimal hyperspherical energy asymptotically corresponds to the uniform distribution on $\mathbb{S}^d$ still holds. Note that, we usually will not have the case of $s > d$ in our applications.

F.3 Experimental Settings

Table F.1: Our plain CNN architectures with different convolutional layers. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers. E.g., [3×3, 64]×3 denotes 3 cascaded convolution layers with 64 filters of size 3×3.

Layer	CNN-6 (CIFAR-100)	CNN-9 (CIFAR-100)	CNN-10 (ImageNet-2012)
Conv1.x	[3×3, 64]×2	[3×3, 64]×3	[7×7, 64], Stride 2 3×3, Max Pooling, Stride 2 [3×3, 64]×3
Pool1	2×2 Max Pooling, Stride 2		
Conv2.x	[3×3, 64]×2	[3×3, 64]×3	[3×3, 128]×3
Pool2	2×2 Max Pooling, Stride 2		
Conv3.x	[3×3, 64]×2	[3×3, 64]×3	[3×3, 256]×3
Pool3	2×2 Max Pooling, Stride 2		
Fully Connected	64	64	256

Table F.2: Our ResNet architectures with different convolutional layers. Conv0.x, Conv1.x, Conv2.x, Conv3.x and Conv4.x denote convolution units that may contain multiple convolutional layers, and residual units are shown in double-column brackets. Conv1.x, Conv2.x and Conv3.x usually operate on different size feature maps. These networks are essentially the same as [61], but some may have a different number of filters in each layer. The downsampling is performed by convolutions with a stride of 2. E.g., [3×3, 64]×4 denotes 4 cascaded convolution layers with 64 filters of size 3×3, S2 denotes stride 2.

Layer	ResNet-20 (CIFAR-100)	ResNet-32 (CIFAR-100)
Conv1.x	[3×3, 16]×1 $\begin{bmatrix} 3\times3, 16 \\ 3\times3, 16 \end{bmatrix} \times 3$	[3×3, 16]×1 $\begin{bmatrix} 3\times3, 16 \\ 3\times3, 16 \end{bmatrix} \times 5$
Conv2.x	$\begin{bmatrix} 3\times3, 32 \\ 3\times3, 32 \end{bmatrix} \times 3$	$\begin{bmatrix} 3\times3, 32 \\ 3\times3, 32 \end{bmatrix} \times 5$
Conv3.x	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 5$
	Average Pooling	

Reported Results. For all the experiments on MLPs and CNNs (except CNNs in the few-shot learning), we report testing error rates. For the few-shot learning experiment, we report testing accuracy. For all the experiments on both GCNs and PointNets, we report testing accuracy. All results are averaged over 10 runs of the model.

Multilayer perceptron. We conduct digit classification task on MNIST with a three-

layer multilayer perceptrons following this repository[1] . The input dimension of each MNIST digit is 28×28, which is 784 dimensions after flattened. Our two hidden layers have 256 output dimensions, *i.e.*, 256 neurons. The output layer will output 10 logits for classification. We use a cross-entropy loss with softmax function. For the optimization, we use a momentum SGD with learning rate 0.01, momentum 0.9 and batch size 100. The training stops in 100 epochs.

Convolutional neural networks. The network architectures used in the paper are elaborated in Table F.1 and Table F.2. For CIFAR-100, we use 128 as the mini-batch size. We use momentum SGD with momentum 0.9 and the learning rate starts with 0.1, divided by 10 when the performance is saturated. For ImageNet-2012, we use batch size 128 and start with learning rate 0.1. The learning rate is divided by 10 when the performance is saturated, and the training is terminated at 700k iterations. For ResNet-20 and ResNet-32 on CIFAR-100, we use exactly the same architecture used on CIFAR-10 as [61]. The rotation matrix is initialized with random normal distribution (mean is 0 and variance is 1). Note that, for all the compared methods, we always use the best possible hyperparameters to make sure that the comparison is fair. The baseline has exactly the same architecture and training settings as the one that OPT uses. If not otherwise specified, standard ℓ_2 weight decay ($5e-4$) is applied to all the neural network including baselines and the networks that use OPT training.

Few-shot learning. The network architecture (Table F.3) we used for few-shot learning experiments is the same as that used in [189]. In our experiments, we show comparison of our OPT training with standard training on 'baseline' and 'baseline++' settings in [189]. In 'baseline' setting, a standard CNN model is pretrained on the whole meta-train dataset (standard non-MAML supervised training) and later only the classifier layer is finetuned on few-shot dataset. 'baseline++' differs from 'baseline' on the classifier: in 'baseline', each output dimension of the classifier is computed as the inner product between weight w

[1] https://github.com/hwalsuklee/tensorflow-mnist-MLP-batch_normalization-weight_initializers

and input x, i.e. $w \cdot x$; while in 'baseline++' it becomes the scaled cosine distance $c\frac{w \cdot x}{\|w\|\|x\|}$ where c is a positive scalar. Following [189], we set $c = 2$.

During pretraining, the model is trained for 200 epochs on the meta-train set of mini-ImageNet with an Adam optimzer (learning rate $1e-3$, weight decay $5e-4$) and the classifier is discarded after pretraining. The model is later finetuned, with a new classifier, on the few-shot samples (5 way, support size 5) with a momentum SGD optimizer (learning rate $1e-2$, momentum 0.9, dampening 0.9, weight decay $1e-3$, batch size 4) for 100 epochs. We re-initialize the classifier for each few-shot sample.

Table F.3: The number of classes is different for pretraining and finetuning.

Layer	CNN-4
Conv1	3×3, 64
Pool1	2×2 Max Pooling, Stride 2
Conv2	3×3, 64
Pool2	2×2 Max Pooling, Stride 2
Conv3	3×3, 64
Pool3	2×2 Max Pooling, Stride 2
Conv4	3×3, 64
Pool4	2×2 Max Pooling, Stride 2
Linear Classifier	number of classes

Graph neural networks. We implement the OPT training for GCN on the official repositories[2]. The experimental settings also follow the official repository to ensure a fair comparison. For OPT (CP) method, we use the original hyperparameters and experimental setup except the added rotation matrix. For OPT (OGD) method, we use our own OGD optimizer in Tensorflow to train the rotation matrix in order to maintain orthogonality and use the original optimizer to train the other variables.

Point cloud recognition. For the PointNet experiments, we exactly follow the same setting in the original paper [158] and the official repositories[3]. Specifically, we multiply the rotation matrix to the original fixed neurons in all the 1×1 convolution layers and the fully connected layer except the final classifier. All the rotation matrix is initialized with random normal distribution. For PointNet experiments, we use point number 1024, batch

[2]https://github.com/tkipf/gcn

[3]https://github.com/charlesq34/pointnet

size 32 and Adam optimizer started with learning rate 0.001, the learning rate will decay by 0.7 every 200k iterations, and the training is terminated at 250 epochs.

F.4 Theoretical Discussion on Optimization and Generalization

The key problem we discuss in this section is why OPT may lead to easier optimization and better generalization. We have already shown that OPT can guarantee the minimum hyperspherical energy (MHE) in a probabilistic sense. Although empirical evidences [31] have shown significant and consistent performance gain by minimizing hyperspherical energy, why lower hyperspherical energy will lead to better generalization is still unclear. We argue that OPT leads to better generalization from two aspects: how OPT may affect the training and generalization, and why minimum hyperspherical energy serves as a good inductive bias. We note that rigorously proving that OPT generalizes better is out of the scope of this paper and remains our future work. The section serves as a very preliminary discussion for this topic, and hopefully the discussion can inspire more theoretical studies about OPT.

Our goal here is to leverage and apply existing theoretical results [166, 126, 206, 207, 187, 186] to explain the role that MHE plays rather than proving sharp and novel generalization bounds. We emphasize that our paper is *NOT* targeted as a theoretical one that proves novel generalization bounds.

We simply consider one-hidden-layer networks as the hypothesis class:

$$\mathcal{F} = \{f(x) = \sum_{j=1}^{n} v_j \sigma(\boldsymbol{w}_j^\top \boldsymbol{x}) : v_j \in \{\pm 1\}, \sum_{j=1}^{n} \|\boldsymbol{w}_j\| \leq C_w\} \tag{F.28}$$

where $\sigma(\cdot) = \max(0, \cdot)$ is ReLU. Since the magnitude of v_j can be scaled into $\boldsymbol{w}_j$, we can restrict v_j to be ± 1. Given a set of *i.i.d.* training sample $\{\boldsymbol{x}_i, y_i\}_{i=1}^m$ where $x \in \mathbb{R}^d$ is drawn uniformly from the unit hypersphere, we minimize the least square loss $\mathcal{L} = \frac{1}{2m} \sum_{i=1}^{m} (y_i -$

$f(\boldsymbol{x}_i))^2$. The gradient *w.r.t.* $\boldsymbol{w}_i$ is

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_j} = \frac{1}{m}\sum_{i=1}^{m}\big(f(\boldsymbol{x}_i) - y_i\big)v_j\sigma'(\boldsymbol{w}_j^\top \boldsymbol{x}_i)\boldsymbol{x}_i. \tag{F.29}$$

Let $\boldsymbol{W} := \{\boldsymbol{w}_1^\top, \cdots, \boldsymbol{w}_n^\top\}^\top$ be the column concatenation of neuron weights. We aim to identify the conditions under which there are no spurious local minima. We rewrite that

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} = \big(\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_1}^\top, \cdots, \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_n}^\top\big)^\top = \boldsymbol{D}\boldsymbol{r} \tag{F.30}$$

where $\boldsymbol{r} \in \mathbb{R}^m$ $\boldsymbol{r}_i = \frac{1}{m}f(\boldsymbol{x}_i) - y_i$, $\boldsymbol{D} \in \mathbb{R}^{n\times m}$, and $\boldsymbol{D}_{ij} = v_i\sigma'(\boldsymbol{w}_i^\top \boldsymbol{x}_j)\boldsymbol{x}_j$. Therefore, we can obtain the following inequality:

$$\|\boldsymbol{r}\| \leq \frac{1}{s_m(\boldsymbol{D})}\left\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}\right\| \tag{F.31}$$

where $\|\boldsymbol{r}\|$ is the training error and $s_m(\boldsymbol{D})$ is the minimum singular value of $\boldsymbol{D}$. If we need the training error to be small, then we have to lower bound $s_m(\boldsymbol{D})$ away from zero. Therefore, the essential problem now becomes the relationship between MHE and the lower bound of $s_m(\boldsymbol{D})$. We have the following result from [126]:

Lemma 14. *With probability larger than* $1 - m\exp(-m\gamma_m/8) - 2m^2\exp(-4\log^2 d) - \delta$, *we will have that* $s_m(\boldsymbol{D})^2 \geq \frac{1}{2}nm\gamma_m - cn\rho(\boldsymbol{W})$ *where*

$$\begin{aligned}\rho(\boldsymbol{W}) \leq \frac{\log d}{\sqrt{d}}\sqrt{2L_2(\boldsymbol{W})}m\big(\frac{4}{m}\log\frac{1}{\delta}\big)^{1/4} \\ + \frac{2\log d}{\sqrt{d}}m\sqrt{\frac{4}{3m}\log\frac{1}{\delta}} + \frac{\log d}{\sqrt{d}}mL_2(\boldsymbol{W}) + 2,\end{aligned} \tag{F.32}$$

and $L_2(\boldsymbol{W}) = \frac{1}{n^2}\sum_{i,j=1}^{n}k(\boldsymbol{w}_i, \boldsymbol{w_j})^2 - \mathbb{E}_{\boldsymbol{u},\boldsymbol{v}}[k(u,v)^2]$. *The kernel function* $k(\boldsymbol{u},\boldsymbol{v})$ *is* $\frac{1}{2} - \frac{1}{2\pi}\arccos(\frac{\langle \boldsymbol{u},\boldsymbol{v}\rangle}{\|\boldsymbol{u}\|\|\boldsymbol{v}\|})$.

Once MHE is achieved, the neurons will be uniformly distributed on the unit hypersphere. From Lemma 14, we can see that if the neurons are uniformly distributed on the

unit hypersphere, $L_2(\boldsymbol{W})$ will be very small and close to zero. Then $\rho(\boldsymbol{W})$ will also be small, leading to large lower bound for $s_m(\boldsymbol{D})$. Therefore, MHE can result in small training error once the gradient norm $\left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} \right\|$ is small. The result implies no spurious local minima if we use OPT for training.

Furthermore, suppose that $\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} \|^2 \leq \epsilon$, [126] also proves a training error bound $\tilde{\mathcal{O}}(\epsilon)$ and a generalization bound bound $\tilde{\mathcal{O}}(\epsilon + \frac{1}{\sqrt{m}})$ based on the assumption that $\boldsymbol{W}$ belongs to a specific set $\mathcal{G}_{\boldsymbol{W}}$ (for the definition of $\mathcal{G}_{\boldsymbol{W}}$, please refer to [126]). Therefore, MHE is also connected to the training and generalization error. Note that, the analysis here is highly simplified and the purpose here is to give some justifications rather than rigorously proving any bound.

We further argue that MHE induced by OPT serves as an important inductive bias for neural networks. As the standard regularizer for neural networks, weight decay controls the norm of the neuron weights, regularizing essentially one dimension of the weight. In contrast, MHE completes an important missing pieces by regularizing the remaining dimensions of the weight. MHE encourages minimum hyperspherical redundancy between neurons. In the linear classifier case, MHE impose a prior of maximal inter-class separability.

F.5 On Parameter-Efficient OPT

F.5.1 Formulation

Since OPT over-parameterizes the neurons, it will consume more GPU memory in training (note that, the number of parameters will not increase in testing). For a d-dimensional neuron, OPT will learn an orthogonal matrix of size $d \times d$ that applies to the the neuron. Therefore, we will need d^2 extra parameters for one layer of neurons, making the training more expensive in terms of the GPU memory. Although the extra training overhead in OPT will not affect the inference speed of the trained neural networks, we still desire to achieve better parameter efficiency in OPT. To this end, we discuss some design possibilities for

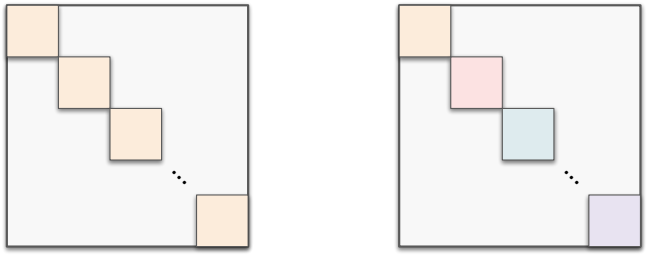


Figure F.1: Comparison between the block-shared matrix $\boldsymbol{R}_s$ and the unconstrained block matrix $\boldsymbol{R}_u$.

the *parameter-efficient OPT* (PE-OPT) in this section.

Original OPT over-parameterize a neuron $\boldsymbol{v} \in \mathbb{R}^{n\times n}$ with $\boldsymbol{R}\boldsymbol{v}$ where $\boldsymbol{R}$ is a layer-shared orthogonal matrix of size $d \times d$. We aim to reduce the effective parameters of this $d \times d$ orthogonal matrix. We incorporate a block-diagonal structure to the orthogonal matrix $\boldsymbol{R}$. Specifically, we formulate $\boldsymbol{R}$ as $\text{Diag}(\boldsymbol{R}^{(1)}, \boldsymbol{R}^{(2)}, \cdots, \boldsymbol{R}^{(k)})$ where $\boldsymbol{R}^{(i)}$ is an orthogonal matrix with size $d_i \times d_i$ (it is easy to see that we need $d = \sum_i d_i$). As an example, we only consider the case where all $\boldsymbol{R}^{(i)}$ are of the same size (*i.e.*, $d_1 = d_2 = \cdots = d_k = \frac{d}{k}$). It is also obvious that as long as each block is an orthogonal matrix, then the overall matrix $\boldsymbol{R}$ remains an orthogonal matrix.

First, we consider that all the block matrices on the diagonal of the orthogonal matrix $\boldsymbol{R}$ are shared, meaning that $\boldsymbol{R} = \text{Diag}(\boldsymbol{R}^{(1)}, \boldsymbol{R}^{(1)}, \cdots, \boldsymbol{R}^{(1)})$ (*i.e.*, $\boldsymbol{R}^{(1)} = \boldsymbol{R}^{(2)} = \cdots = \boldsymbol{R}^{(k)}$). Therefore, we have a block-diagonal matrix $\boldsymbol{R}_s$ with shared block $\boldsymbol{R}^{(1)}$ as the final orthogonal matrix for the neuron $\boldsymbol{v}$:

$$\boldsymbol{R}_s = \begin{bmatrix} \boldsymbol{R}^{(1)} & 0 & \cdots & 0 \\ 0 & \boldsymbol{R}^{(1)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \boldsymbol{R}^{(1)} \end{bmatrix} \tag{F.33}$$

where $\boldsymbol{R}^{(1)} \in \mathbb{R}^{\frac{d}{k}\times\frac{d}{k}}$. The effective number of parameters for the orthogonal matrix $\boldsymbol{R}_s$ immediately reduces to $\frac{d^2}{k^2}$. The left figure in Figure F.1 gives an intuitive illustration for the

block-shared matrix $\boldsymbol{R}_s$. Therefore, PE-OPT only needs to learn $\boldsymbol{R}^{(1)}$ in order to construct the orthogonal matrix of size $d \times d$.

Second, we consider that all the diagonal block matrices are independent, indicating that $\boldsymbol{R} = \text{Diag}(\boldsymbol{R}^{(1)}, \boldsymbol{R}^{(2)}, \cdots, \boldsymbol{R}^{(k)})$ where $\boldsymbol{R}^{(i)}, \forall i$ are different orthogonal matrices in general. We term such matrix $\boldsymbol{R}$ as unconstrained block matrix. Therefore, we have the unconstrained block diagonal matrix $\boldsymbol{R}_u$ as

$$\boldsymbol{R}_u = \begin{bmatrix} \boldsymbol{R}^{(1)} & 0 & \cdots & 0 \\ 0 & \boldsymbol{R}^{(2)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \boldsymbol{R}^{(k)} \end{bmatrix} \tag{F.34}$$

where the orthogonal matrices $\boldsymbol{R}^{(i)}, \forall i$ will be learned independently. The effective number of parameters for the orthogonal matrix $\boldsymbol{R}_u$ is $\frac{d^2}{k}$, making it more flexible than the block-shared matrix $\boldsymbol{R}_s$.

Let's consider a convolution neuron (*i.e.*, convolution filter) $\boldsymbol{v} \in \mathbb{R}^{c_1 \times c_2 \times c_3}$ (*e.g.*, a typical convolution neuron is of size $3 \times 3 \times 64$) as an example. The orthogonal matrix $\boldsymbol{R}$ for the convolution neuron is of size $(c_1 c_2 c_3) \times (c_1 c_2 c_3)$. Typically, we will divide the neuron into k sub-neuron along the c_3-axis, each with size $c_1 \times c_2 \times \frac{c_3}{k}$. Then in order to learn a block-shared orthogonal matrix $\boldsymbol{R}_s$, we will essentially learn a shared orthogonal matrix of size $(\frac{1}{k} c_1 c_2 c_3) \times (\frac{1}{k} c_1 c_2 c_3)$ that applies to each sub-neuron (there are k sub-neurons of size $c_1 \times c_2 \times \frac{c_3}{k}$ in total). For the case of learning a unconstrained block-diagonal orthogonal matrix $\boldsymbol{R}_u$, we simply learn different orthogonal matrices for different sub-neurons.

F.5.2 Experiments and Results

We conduct the image recognition experiments on CIFAR-100 with CNN-6 described in Table F.1. The setting is exactly the same as subsection 7.5.3. For the convolution filter, we use the size of $3 \times 3 \times 64$, *i.e.*, $c_1 = 3, c_2 = 3, c_3 = 64$. The results are given in Table F.4

and Table F.5. "# Parameters" in both tables denote the number of effective parameters for the orthogonal matrix $\boldsymbol{R}$ in a single layer. The baseline with fixed neurons is only to train the final classifiers with the randomly initialized neuron weights staying fixed. It means that this baseline basically removes the learnable orthogonal matrices but still fixes the neuron weights, so it only achieves 73.81% testing error. As expected, as the number of effective parameters goes down, the performance of PE-OPT generally decreases. One can also observe that using separate orthogonal matrices generally yields better performance than shared orthogonal matrices. $k = 2$ and $k = 4$ seems to be a reasonable trade-off between better accuracy and less parameters.

When k becomes larger (*i.e.*, the number of parameters become less) in the case of block-shared orthogonal matrices, we find that PE-OPT (LS) performs the best among all the variants. When k becomes larger (*i.e.*, the number of parameters become less) in the case of unconstrained block orthogonal matrices, we can see that both PE-OPT (GS) and PE-OPT (LS) performs better than the other variants.

Table F.4: Testing error (%) on CIFAR-100 with different settings of PE-OPT (with block-shared orthogonal matrix $\boldsymbol{R}_s$).

Method	# Parameters	PE-OPT (CP)	PE-OPT (GS)	PE-OPT (HR)	PE-OPT (LS)	PE-OPT (OGD)
$c_3/k = 64$ $(k = 1)$ (*i.e.*, Original OPT)	331.7K	33.53	**33.02**	35.67	34.48	33.33
$c_3/k = 32$ $(k = 2)$	82.9K	34.93	**34.39**	35.83	34.50	35.06
$c_3/k = 16$ $(k = 4)$	20.7K	39.40	39.13	39.67	**37.58**	39.80
$c_3/k = 8$ $(k = 8)$	5.2K	47.77	46.65	46.69	**45.62**	47.43
$c_3/k = 4$ $(k = 16)$	1.3K	56.65	55.91	55.69	**54.75**	57.15
$c_3/k = 2$ $(k = 32)$	0.3K	63.46	62.65	62.38	**61.60**	62.46
$c_3/k = 1$ $(k = 64)$	0.1K	67.36	67.11	67.05	**66.61**	67.23
Baseline	-			37.59		
Baseline with fixed random neurons	-			73.81		

Table F.5: Testing error (%) on CIFAR-100 with different settings of PE-OPT (with unconstrained block orthogonal matrix $\boldsymbol{R}_u$).

Method	# Parameters	PE-OPT (CP)	PE-OPT (GS)	PE-OPT (HR)	PE-OPT (LS)	PE-OPT (OGD)
$c_3/k = 64$ $(k = 1)$ (*i.e.*, Original OPT)	331.7K	33.53	**33.02**	35.67	34.48	33.33
$c_3/k = 32$ $(k = 2)$	165.9K	33.54	**33.15**	35.65	34.09	34.27
$c_3/k = 16$ $(k = 4)$	82.9K	34.77	**34.50**	35.71	34.96	35.97
$c_3/k = 8$ $(k = 8)$	41.5K	37.25	36.43	36.40	**36.17**	39.75
$c_3/k = 4$ $(k = 16)$	20.7K	40.74	**39.89**	39.98	39.93	43.43
$c_3/k = 2$ $(k = 32)$	10.4K	45.36	44.77	44.83	**44.61**	48.98
$c_3/k = 1$ $(k = 64)$	5.2K	50.94	**49.16**	49.57	49.23	54.93
Baseline	-			37.59		
Baseline with fixed random neurons	-			73.81		

F.6 On Generalizing OPT: Over-Parameterized Training with Constraint

OPT opens many new possibilities in training neural networks. We consider a simple generalization to OPT in this section to showcase the great potential of OPT. Instead of constraining the over-parameterization matrix $\boldsymbol{R} \in \mathbb{R}^{d\times d}$ in Equation 7.1 to be orthogonal, we can use any meaningful structural constraints for this matrix, and even regularize it in a task-driven way. Furthermore, instead of a linear over-parameterization (*i.e.*, multiplying a matrix $\boldsymbol{R}$) to the neuron, we can also consider nonlinear mapping. We come up with the following straightforward generalization to OPT (the settings and notations exactly follow Equation 7.1):

$$\begin{aligned}
\text{Standard: } & \min_{\boldsymbol{v}_i,u_i,\forall i} \sum_{j=1}^{m} \mathcal{L}\big(y, \sum_{i=1}^{n} u_i \boldsymbol{v}_i^\top \boldsymbol{x}_j\big) \\
\text{Original OPT: } & \min_{\boldsymbol{R},u_i,\forall i} \sum_{j=1}^{m} \mathcal{L}\big(y, \sum_{i=1}^{n} u_i (\boldsymbol{R}\boldsymbol{v}_i)^\top \boldsymbol{x}_j\big) \\
& \text{s.t. } \boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I} \\
\text{Generalized OPT: } & \min_{\boldsymbol{R},u_i,\forall i} \sum_{j=1}^{m} \mathcal{L}\big(y, \sum_{i=1}^{n} u_i (\mathcal{T}(\boldsymbol{v}_i))^\top \boldsymbol{x}_j\big) \\
& \text{s.t. Some constraints on } \mathcal{T}(\cdot)
\end{aligned} \tag{F.35}$$

where $\mathcal{T}(\cdot) : \mathbb{R}^d \to \mathbb{R}^d$ denotes some transformation (including both linear and nonlinear). Notice that the generalized OPT (G-OPT) no longer requires orthogonality. Such formulation of G-OPT can immediately inspire a number of instances. We will discuss some obvious ones here.

If we consider $\mathcal{T}(\cdot)$ to be a linear mapping, we may constrain $\boldsymbol{R}$ to be symmetric positive definite other than orthogonal. A simple way to achieve that is to use Cholesky factorization $\boldsymbol{L}\boldsymbol{L}^\top$ where $\boldsymbol{L}$ is a lower triangular matrix to parameterize the matrix $\boldsymbol{R}$. Essentially, we learn a lower triangular matrix $\boldsymbol{L}$ and use $\boldsymbol{L}\boldsymbol{L}^\top$ to replace $\boldsymbol{R}$ in OPT. The positive definiteness provides the transformation $\boldsymbol{R}$ with some geometric constraint. Specifically, a

positive definite $\boldsymbol{R}$ only transforms the neuron weight $\boldsymbol{v}$ to the direction that has the angle less than $\frac{\pi}{2}$ to $\boldsymbol{v}$, because $\boldsymbol{v}^\top \boldsymbol{R}\boldsymbol{v} > 0$. Moreover, we can also require the transformation to have structural constraints on $\boldsymbol{R}$. For example, $\boldsymbol{R}$ can be upper (lower) triangular, banded, symmetric, skew-symmetric, upper (lower) Hessenberg, etc.

We can also consider $\mathcal{T}(\cdot)$ to be a nonlinear mapping. A obvious example is to use a neural network (*e.g.*, MLP, CNN) as $\mathcal{T}(\cdot)$. Then the nonlinear G-OPT will share some similarities with HyperNetworks [208] and Network-in-Network [188]. If we further consider $\mathcal{T}(\cdot)$ to be dependent on the input, then the nonlinear G-OPT will have close connections to dynamic neural networks [209, 33].

To summarize, OPT provides a novel and effective framework to train neural networks and may inspire many different threads of future research.

F.7 Hyperspherical Energy Training Dynamics of Individual Layers

We also plot the hyperspherical energy ($\boldsymbol{E}(\hat{\boldsymbol{v}}_i|_{i=1}^n) = \sum_{i=1}^n \sum_{j=1,j\neq i}^n \|\hat{\boldsymbol{v}}_i - \hat{\boldsymbol{v}}_j\|^{-1}$ in which $\hat{\boldsymbol{v}}_i = \frac{\boldsymbol{v}_i}{\|\boldsymbol{v}_i\|}$ is the i-th neuron weight projected onto the unit hypersphere.) in every layer of CNN-6 during training to show how these hyperspherical energies are being minimized. From Figure F.2, we can observe that OPT can always maintain the minimum hyperspherical energy during the entire training process, while the MHE regularization cannot. Moreover, the hyperspherical energy of the baseline will also decrease as the training proceeds, but it is still much higher than the OPT training.

F.8 More Discussions

Flexible training. First, OPT can used in multi-task training [210] where each set of orthogonal matrices represent one task. OPT can learn different set of orthogonal matrices for different tasks with the neuron weights remain the same. Second, we can perform progressive training with OPT. For example, after learning a set of orthogonal matrices on a large coarse-grained dataset (*i.e.*, pretraining), we can multiple the orthogonal matrices

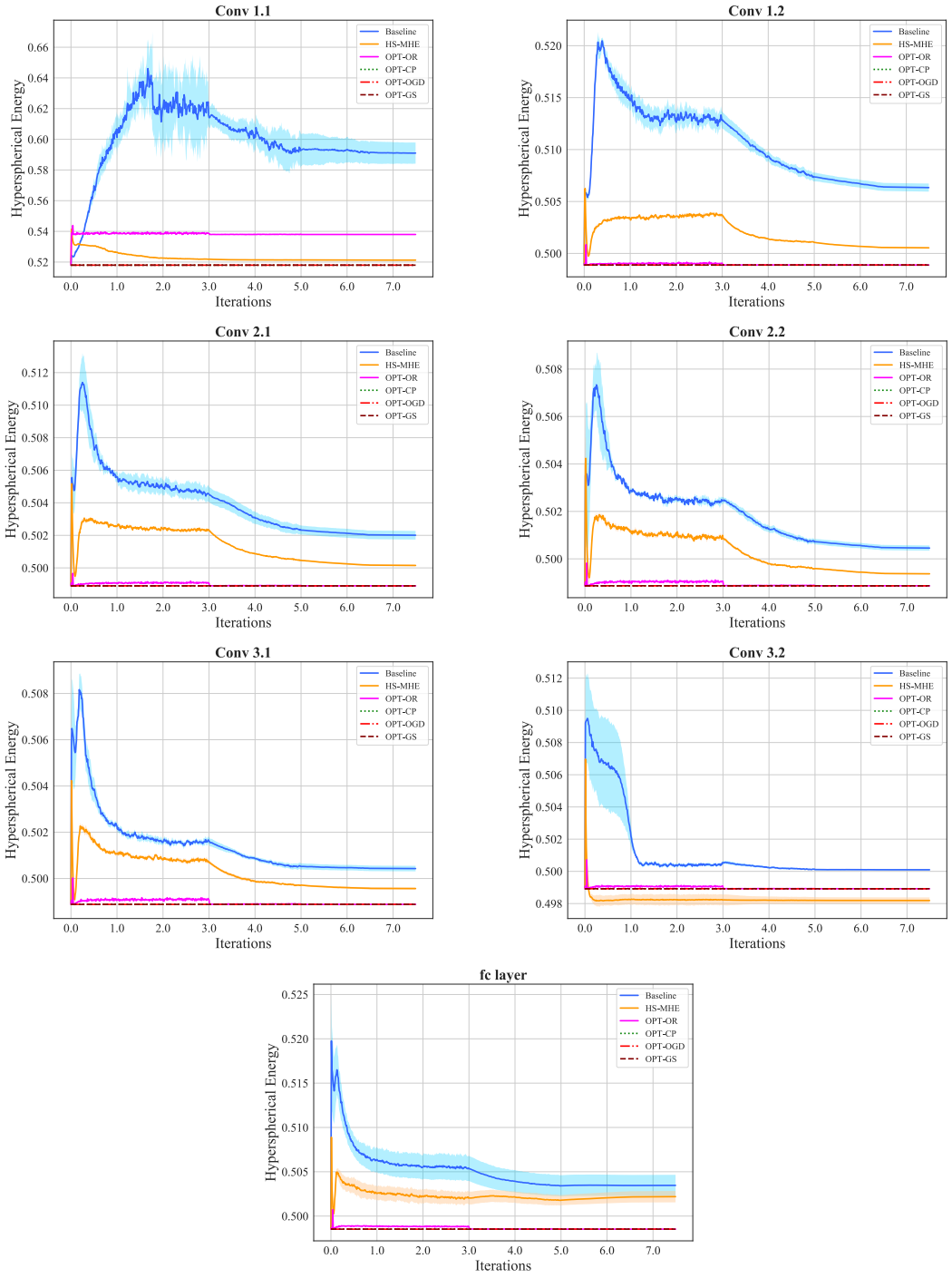


Figure F.2: Training dynamics of hyperspherical energy in each layer of CNN-6. We average results with 10 runs.

back to the neuron weights and construct a new set of neuron weights. Then we can use the new neuron weights as a starting point and apply OPT to train on a small fine-grained

dataset (*i.e.*, finetuning).

Limitations and open problems The limitations of OPT include more GPU memory consumption and heavy computation during training, more numerical issues when ensuring orthogonality and weak scalability for ultra wide neural networks. Therefore, there will be plenty of open problems in OPT, such as scalable and efficient training. Most significantly, OPT opens up a new possibility for studying theoretical generalization of deep networks. With the decomposition to hyperspherical energy and coordinate system, OPT provides a new perspective for future research.

F.9 Geometric Properties of Randomly Initialized Neurons

There are many interesting geometric properties [211, 212] of random points distributed independently and uniformly on the unit hypersphere. We summarize a few of them that make randomly initialized neurons distinct from any deterministic neuron configuration. Note that, there exist many deterministic neuron configurations that can also achieve very low hyperspherical energy, and this section aims to describe a few unique geometric properties of randomly initialized neurons.

There are two widely used geometric properties corresponding to a neuron configuration (*i.e.*, a set of neurons) $\hat{\boldsymbol{W}}_N = \{\hat{\boldsymbol{w}}_1, \cdots, \hat{\boldsymbol{w}}_N \in \mathbb{S}^d\}$. In the main paper, we define neurons on $\mathbb{S}^{d-1}$, but without loss of generality we define neurons on $\mathbb{S}^d$ here for convenience. The first one is the *covering radius*:

$$\alpha(\hat{\boldsymbol{W}}_N) := \alpha(\hat{\boldsymbol{W}}_N; \mathbb{S}^{d-1}) := \max_{\boldsymbol{u}\in\mathbb{S}^d} \min_{1\leq i\leq N} \arccos(\boldsymbol{u}, \hat{\boldsymbol{w}}_i) \tag{F.36}$$

which is the biggest geodesic distance from a neuron in $\mathbb{S}^d$ to the nearest point in $\hat{\boldsymbol{W}}_N$. The second one is the *separation distance*:

$$\psi(\hat{\boldsymbol{W}}_N) := \min_{1\leq i,j,\leq N, i\neq j} \arccos(\hat{\boldsymbol{w}}_i, \hat{\boldsymbol{w}}_j) \tag{F.37}$$

which gives the least geodesic distance between arbitrary two points in $\hat{\boldsymbol{W}}_N$. Random points (*i.e.*, randomly initialized neurons) typically have poor separation properties, since the separation is sensitive to the specific placement of points. [211] shows an example on $\mathbb{S}^1$ to illustrate this observation.

[211] considers a different but related quantity, *i.e.*, the sume of powers of the "hole radii". A set of neurons $\hat{\boldsymbol{W}}_N$ on $\mathbb{S}^d$ uniquely defines a convex polytope, which can be viewed as the convex hull of the neuron configuration. Each facet of the polytope defines a "hole". Such a hole denotes the maximal spherical cap for a facet that contains neurons of $\hat{\boldsymbol{W}}_N$ only on the boundary. It is easy to see that the geodesic radius of the largest hole is the covering radius $\alpha(\hat{\boldsymbol{W}}_N)$. We assume that for the set of neurons $\hat{\boldsymbol{W}}_N$, there are f_d holes (*i.e.*, facets) in total. Therefore, the i-th hole radius is defined as $\rho_i = \rho_i(\hat{\boldsymbol{W}}_N)$ which is the Euclidean distance in $\mathbb{R}^{d+1}$ between the center of the i-th spherical cap and the boundary. The i-th spherical cap is located on the sphere corresponding to the i-th facet. We have that $\rho_i = 2\sin(\frac{\alpha_i}{2})$ where α_i is the geodesic radius of the i-th spherical cap. We are interested in the sums of the p-th powers of the hole radii, *i.e.*,

$$\mathcal{P} = \sum_{i=1}^{f_d} (\rho_i)^p \tag{F.38}$$

where p is larger than zero. For large p, the largest hole dominates:

$$\lim_{p\to\infty} (\mathcal{P})^{\frac{1}{p}} = \lim_{p\to\infty} \left(\sum_{i=1}^{f_d} (\rho_i)^p \right)^{\frac{1}{p}} = \max_{1\le i\le f_d} \rho_i = 2\sin(\frac{\alpha(\hat{\boldsymbol{W}}_N)}{2}) \tag{F.39}$$

where $\rho(\hat{\boldsymbol{W}}_N) := \max_{1\le i\le f_d} \rho_i$. Then we introduce some useful notations to state the geometric properties. Let ψ_d be the surface area of $\mathbb{S}^d$, and we have that

$$\psi_d = \frac{2\pi^{\frac{d+1}{2}}}{\Gamma(\frac{d+1}{2})}, \tag{F.40}$$

and we also define the following quantities (with $\psi_0 = 2$):

$$\begin{aligned} \kappa_d &:= \frac{1}{d}\frac{\psi_{d-1}}{\psi_d} = \frac{1}{d}\frac{\Gamma(\frac{d+1}{2})}{\sqrt{\pi}\Gamma(\frac{d}{2})} \\ B_d &:= \frac{2}{d+1}\frac{\kappa_{d^2}}{(\kappa_d)^d} \end{aligned} \tag{F.41}$$

where κ_d can be alternatively defined with the recursion: $\kappa_1 = \frac{1}{\pi}$ and $\kappa_d = \frac{1}{2\pi d\kappa_{d-1}}$. [213] gives the expected number of facets constructed from N random neurons that are independently and uniformly distributed on the unit hypersphere $\mathbb{S}^d$:

$$\mathbb{E}[f_d] = B_d N\big(1 + o(1)\big) \tag{F.42}$$

where $N \to \infty$. Then we introduce the main results of [211] (asymptotics for the expected moments of the hole radii) in the following lemma:

Lemma 15. *If $p \geq 0$ and $\hat{\boldsymbol{w}}_1, \cdots \hat{\boldsymbol{w}}_N$ are N neurons on $\mathbb{S}^d$ that are independently and randomly distributed with respect to the normalized surface area measure σ_d on $\mathbb{S}^d$, then we have that*

$$\begin{aligned} \mathbb{E}[\mathcal{P}] &= B_d(\kappa_d)^{-\frac{p}{d}}\frac{\Gamma(d+\frac{p}{d})\Gamma(N+1)}{\Gamma(d)\Gamma(N+\frac{p}{d})}\big(1+\mathcal{O}(N^{-\frac{2}{d}})\big) \\ &= c_{d,p}N^{1-\frac{p}{d}}\big(1+\mathcal{O}(N^{-\frac{2}{d}})\big) \end{aligned} \tag{F.43}$$

as $N \to \infty$, where $\rho_i = \rho_{i,N}$ is the Euclidean hole radius associated with the i-th facet of the convex hull of $\hat{\boldsymbol{W}}_N$, $c_{d,p} := B_d B_{d,p}$, and $B_{d,p} := \frac{\Gamma(d+\frac{p}{d})}{\Gamma(d)}(\kappa_d)^{-\frac{p}{d}}$. The $\mathcal{O}$-terms above depend on d and p.

As we mentioned, there are many deterministic point (*i.e.*, neuron) configurations such as minimizing hyperspherical energy (*i.e.*, Riesz s-energy) [31] (as $s \to \infty$, the minimal s-energy points approach the best separation), maximizing the determinant for polynomial interpolation [214], Fibonacci points, spherical t-designs, minimizing covering radius (*i.e.*, best covering problem), maximizing the separation (*i.e.*, best packing problem) and maximizing the s-polarization, etc. We note that randomly initialized neurons are quite differ-

ent from these deterministic neuron configurations and have unique geometric properties.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[3] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[5] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[6] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do imagenet classifiers generalize to imagenet?" *arXiv preprint arXiv:1902.10811*, 2019.

[7] B. Chen, W. Liu, A. Garg, Z. Yu, A. Shrivastava, J. Kautz, and A. Anandkumar, "Angular visual hardness," *arXiv preprint arXiv:1912.02279*, 2019.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] W. Liu, Y. Wen, Z. Yu, and M. Yang, "Large-margin softmax loss for convolutional neural networks," in *ICML*, 2016.

[10] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "Sphereface: Deep hypersphere embedding for face recognition," in *CVPR*, 2017.

[11] F. Wang, X. Xiang, J. Cheng, and A. L. Yuille, "Normface: L2 hypersphere embedding for face verification," *arXiv preprint arXiv:1704.06369*, 2017.

[12] W. Liu, Y.-M. Zhang, X. Li, Z. Yu, B. Dai, T. Zhao, and L. Song, "Deep hyperspherical learning," in *NeurIPS*, 2017.

[13] H. Wang, Y. Wang, Z. Zhou, X. Ji, Z. Li, D. Gong, J. Zhou, and W. Liu, "Cosface: Large margin cosine loss for deep face recognition," *arXiv preprint arXiv:1801.09414*, 2018.

[14] F. Wang, W. Liu, H. Liu, and J. Cheng, "Additive margin softmax for face verification," *arXiv preprint arXiv:1801.05599*, 2018.

[15] J. Deng, J. Guo, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," *arXiv preprint arXiv:1801.07698*, 2018.

[16] R. Ranjan, C. D. Castillo, and R. Chellappa, "L2-constrained softmax loss for discriminative face verification," *arXiv preprint arXiv:1703.09507*, 2017.

[17] Y. Zheng, D. K. Pal, and M. Savvides, "Ring loss: Convex feature normalization for face recognition," in *CVPR*, 2018.

[18] Y. Liu, H. Li, and X. Wang, "Rethinking feature discrimination and polymerization for large-scale recognition," *arXiv preprint arXiv:1710.00870*, 2017.

[19] X. Zhang, Z. Fang, Y. Wen, Z. Li, and Y. Qiao, "Range loss for deep face recognition with long-tailed training data," in *ICCV*, 2017.

[20] W. Liu, Z. Liu, Z. Yu, B. Dai, R. Lin, Y. Wang, J. M. Rehg, and L. Song, "Decoupled networks," *CVPR*, 2018.

[21] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[22] P. Mettes, E. van der Pol, and C. Snoek, "Hyperspherical prototype networks," in *NeurIPS*, 2019.

[23] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *NeurIPS*, 2017.

[24] S. W. Park and J. Kwon, "Sphere generative adversarial network based on geometric moment matching," in *CVPR*, 2019.

[25] T. R. Davidson, L. Falorsi, N. De Cao, T. Kipf, and J. M. Tomczak, "Hyperspherical variational auto-encoders," *arXiv preprint arXiv:1804.00891*, 2018.

[26] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra, "Reducing overfitting in deep networks by decorrelating representations," in *ICLR*, 2016.

[27] P. Rodriguez, J. Gonzalez, G. Cucurull, J. M. Gonfaus, and X. Roca, "Regularizing cnns with locally constrained decorrelations," in *ICLR*, 2017.

[28] D. Xie, J. Xiong, and S. Pu, "All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation," *arXiv:1703.01827*, 2017.

[29] N. Bansal, X. Chen, and Z. Wang, "Can we gain more from orthogonality regularizations in training deep networks?" In *NeurIPS*, 2018.

[30] D. Mishkin and J. Matas, "All you need is a good init," in *ICLR*, 2016.

[31] W. Liu, R. Lin, Z. Liu, L. Liu, Z. Yu, B. Dai, and L. Song, "Learning towards minimum hyperspherical energy," *NeurIPS*, 2018.

[32] R. Lin, W. Liu, Z. Liu, C. Feng, Z. Yu, J. M. Rehg, L. Xiong, and L. Song, "Regularizing neural networks via minimizing hyperspherical energy," in *CVPR*, 2020.

[33] W. Liu, Z. Liu, J. M. Rehg, and L. Song, "Neural similarity learning," in *NeurIPS*, 2019.

[34] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *CVPR*, 2015.

[35] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin, "Deep metric learning with angular loss," in *ICCV*, 2017.

[36] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," in *CVPR*, 2014.

[37] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *CVPR*, 2006.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015.

[39] Y. LeCun, C. Cortes, and C. J. Burges, *The mnist database of handwritten digits*, 1998.

[40] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Technical Report*, 2009.

[41] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," Technical Report, Tech. Rep., 2007.

[42] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[43] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv:1408.5093*, 2014.

[44] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.

[45] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" In *ICCV*, 2009.

[46] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *ICML*, 2013.

[47] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *ICLR*, 2015.

[48] M. Lin, Q. Chen, and S. Yan, "Network in network," in *ICLR*, 2014.

[49] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *arXiv:1302.4389*, 2013.

[50] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *AISTATS*, 2015.

[51] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *CVPR*, 2015.

[52] C.-Y. Lee, P. W. Gallagher, and Z. Tu, "Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree," in *AISTATS*, 2016.

[53] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity," in *ICLR*, 2015.

[54] M. F. Stollenga, J. Masci, F. Gomez, and J. Schmidhuber, "Deep networks with internal selective attention through feedback connections," in *NeurIPS*, 2014.

[55] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *BMVC*, 2015.

[56] Y. Sun, X. Wang, and X. Tang, "Deeply learned face representations are sparse, selective, and robust," in *CVPR*, 2015.

[57] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," *arXiv:1411.7923*, 2014.

[58] C. Ding and D. Tao, "Robust face recognition via multimodal deep face representation," *IEEE TMM*, vol. 17, no. 11, pp. 2049–2058, 2015.

[59] A. Asthana, S. Zafeiriou, S. Cheng, and M. Pantic, "Incremental face alignment in the wild," in *CVPR*, 2014.

[60] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[61] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[62] Y. Sun, Y. Chen, X. Wang, and X. Tang, "Deep learning face representation by joint identification-verification," in *NeurIPS*, 2014.

[63] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *ECCV*, 2016.

[64] G. B. Huang and E. Learned-Miller, "Labeled faces in the wild: Updates and new reporting procedures," *Dept. Comput. Sci., Univ. Massachusetts Amherst, Amherst, MA, USA, Tech. Rep*, pp. 14–003, 2014.

[65] I. Kemelmacher-Shlizerman, S. M. Seitz, D. Miller, and E. Brossard, "The megaface benchmark: 1 million faces for recognition at scale," in *CVPR*, 2016.

[66] A. Ross and A. K. Jain, "Multimodal biometrics: An overview," in *12th European Signal Processing Conference*, IEEE, 2004, pp. 1221–1224.

[67] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *CVPR*, 2014.

[68] Y. Sun, X. Wang, and X. Tang, "Sparsifying neural network connections for face recognition," in *CVPR*, 2016.

[69] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *CVPR*, 2006.

[70] K.-C. Lee, J. Ho, M.-H. Yang, and D. Kriegman, "Video-based face recognition using probabilistic appearance manifolds," in *CVPR*, 2003.

[71] X. He, S. Yan, Y. Hu, P. Niyogi, and H.-J. Zhang, "Face recognition using laplacianfaces," *TPAMI*, vol. 27, no. 3, pp. 328–340, 2005.

[72] A. Talwalkar, S. Kumar, and H. Rowley, "Large-scale manifold learning," in *CVPR*, 2008.

[73] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[74] J. Liu, Y. Deng, and C. Huang, "Targeting ultimate accuracy: Face recognition via deep embedding," *arXiv:1506.07310*, 2015.

[75] L. Wolf, T. Hassner, and I. Maoz, "Face recognition in unconstrained videos with matched background similarity," in *CVPR*, 2011.

[76] D. Miller, I. Kemelmacher-Shlizerman, and S. M. Seitz, "Megaface: A million faces for recognition at scale," *arXiv:1505.02108*, 2015.

[77] H.-W. Ng and S. Winkler, "A data-driven approach to cleaning large face datasets," in *ICIP*, 2014.

[78] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014.

[79] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," in *ICLR*, 2015.

[80] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *NeurIPS*, 2016.

[81] D. Mishkin and J. Matas, "All you need is a good init," *arXiv:1511.06422*, 2015.

[82] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.," in *Aistats*, 2010.

[83] X. Li, Z. Wang, J. Lu, R. Arora, J. Haupt, H. Liu, and T. Zhao, "Symmetry, saddle points, and global geometry of nonconvex matrix factorization," *arXiv:1612.09296*, 2016.

[84] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *IJCV*, pp. 1–42, 2014.

[85] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV*, 2016.

[86] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv:1511.07289*, 2015.

[87] R. Girshick, "Fast r-cnn," in *ICCV*, 2015.

[88] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.

[89] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017.

[90] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.

[91] Y. Yuan, K. Yang, and C. Zhang, "Feature incay for representation regularization," *arXiv preprint arXiv:1705.10284*, 2017.

[92] M. Jones and H. Kobori, "Improving face verification and person re-identification accuracy using hyperplane similarity," in *ICCV*, 2017.

[93] M. Hein and M. Andriushchenko, "Formal guarantees on the robustness of a classifier against adversarial manipulation," in *NIPS*, 2017.

[94] H. N. Mhaskar and C. A. Micchelli, "How to choose an activation function," in *NIPS*, 1994.

[95] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[96] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[97] F. Tramèr, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[98] A. RoyChowdhury, P. Sharma, E. Learned-Miller, and A. Roy, "Reducing duplicate filters in deep neural networks," in *NeurIPS workshop on Deep Learning: Bridging Theory and Practice*, 2017.

[99] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and improving convolutional neural networks via concatenated rectified linear units," in *ICML*, 2016.

[100] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[101] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, "Net-trim: A layer-wise convex pruning of deep neural networks," in *NeurIPS*, 2017.

[102] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[103] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[104] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *arXiv preprint arXiv:1707.01083*, 2017.

[105] P. Xie, A. Singh, and E. P. Xing, "Uncorrelation and evenness: A new diversity-promoting regularizer," in *ICML*, 2017.

[106] P. Xie, Y. Deng, Y. Zhou, A. Kumar, Y. Yu, J. Zou, and E. P. Xing, "Learning latent space models with angular constraints," in *ICML*, 2017.

[107] P. Xie, J. Zhu, and E. Xing, "Diversity-promoting bayesian learning of latent variable models," in *ICML*, 2016.

[108] J. J. Thomson, "Xxiv. on the structure of the atom: An investigation of the stability and periods of oscillation of a number of corpuscles arranged at equal intervals around the circumference of a circle; with application of the results to the theory of atomic structure," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 7, no. 39, pp. 237–265, 1904.

[109] S. Smale, "Mathematical problems for the next century," *The mathematical intelligencer*, vol. 20, no. 2, pp. 7–15, 1998.

[110] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *ICML*, 2009.

[111] I. Ramirez, P. Sprechmann, and G. Sapiro, "Classification and clustering via dictionary learning with structured incoherence and shared features," in *CVPR*, 2010.

[112] N. Li, Y. Yu, and Z.-H. Zhou, "Diversity regularized ensemble pruning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2012.

[113] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine learning*, vol. 51, no. 2, pp. 181–207, 2003.

[114] L. Jiang, D. Meng, S.-I. Yu, Z. Lan, S. Shan, and A. Hauptmann, "Self-paced learning with diversity," in *NeurIPS*, 2014.

[115] P. Xie, W. Wu, Y. Zhu, and E. P. Xing, "Orthogonality-promoting distance metric learning: Convex relaxation and theoretical analysis," in *ICML*, 2018.

[116] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, no. 1, pp. 1929–1958, 2014.

[117] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Neural photo editing with introspective adversarial networks," in *ICLR*, 2017.

[118] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *ICLR*, 2018.

[119] P. M. L. Tammes, "On the origin of number and arrangement of the places of exit on the surface of pollen-grains," *Recueil des travaux botaniques neerlandais*, vol. 27, no. 1, pp. 1–84, 1930.

[120] E. B. Saff and A. B. Kuijlaars, "Distributing many points on a sphere," *The mathematical intelligencer*, vol. 19, no. 1, pp. 5–11, 1997.

[121] A. Kuijlaars and E. Saff, "Asymptotics for minimal discrete energy on the sphere," *Transactions of the American Mathematical Society*, vol. 350, no. 2, pp. 523–538, 1998.

[122] D. Hardin and E. Saff, "Discretizing manifolds via minimum energy points," *Notices of the AMS*, vol. 51, no. 10, pp. 1186–1194, 2004.

[123] N. S. Landkof, *Foundations of modern potential theory*. Springer, 1972, vol. 180.

[124] D. Hardin and E. Saff, "Minimal riesz energy point configurations for rectifiable d-dimensional manifolds," *arXiv preprint math-ph/0311024*, 2003.

[125] M. Götz and E. B. Saff, "Note on d—extremal configurations for the sphere in r d+1," in *Recent Progress in Multivariate Approximation*, Springer, 2001, pp. 159–162.

[126] B. Xie, Y. Liang, and L. Song, "Diverse neural network learns true target functions," *arXiv preprint arXiv:1611.03131*, 2016.

[127] D. Bilyk and M. T. Lacey, "One-bit sensing, discrepancy and stolarskys principle," *Sbornik: Mathematics*, vol. 208, no. 6, p. 744, 2017.

[128] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *NeurIPS*, 2016.

[129] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv preprint arXiv:1312.6120*, 2013.

[130] P. Rodriguez, J. Gonzalez, G. Cucurull, J. M. Gonfaus, and X. Roca, "Regularizing cnns with locally constrained decorrelations," *arXiv preprint arXiv:1611.01967*, 2016.

[131] L. Huang, X. Liu, B. Lang, A. W. Yu, Y. Wang, and B. Li, "Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks," in *AAAI*, 2018.

[132] J. Batle, A. Bagdasaryan, M. Abdel-Aty, and S. Abdalla, "Generalized thomson problem in arbitrary dimensions and non-euclidean geometries," *Physica A: Statistical Mechanics and its Applications*, vol. 451, pp. 237–250, 2016.

[133] M. Calef, W. Griffiths, and A. Schulz, "Estimating the number of stable configurations for the generalized thomson problem," *Journal of Statistical Physics*, vol. 160, no. 1, pp. 239–253, 2015.

[134] Y. Xiang, D. Sun, W. Fan, and X. Gong, "Generalized simulated annealing algorithm and its application to the thomson model," *Physics Letters A*, vol. 233, no. 3, pp. 216–220, 1997.

[135] Y. Xiang and X. Gong, "Efficiency of generalized simulated annealing," *Physical Review E*, vol. 62, no. 3, p. 4473, 2000.

[136] K. Kawaguchi, B. Xie, and L. Song, "Deep semi-random features for nonlinear function approximation," in *AAAI*, 2018.

[137] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NeurIPS*, 2008.

[138] Z. Nehari, *Conformal mapping*. Courier Corporation, 2012.

[139] H. Cramer, *Mathematical methods of statistics (PMS-9)*. Princeton university press, 2016, vol. 9.

[140] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.

[141] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[142] B. Dai, H. Dai, N. He, W. Liu, Z. Liu, J. Chen, L. Xiao, and L. Song, "Coupled variational bayes via optimization embedding," in *NeurIPS*, 2018.

[143] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NeurIPS*, 2014.

[144] Y. Wu and K. He, "Group normalization," in *ECCV*, 2018.

[145] S. Dasgupta and A. Gupta, "An elementary proof of a theorem of johnson and lindenstrauss," *Random Structures & Algorithms*, vol. 22, no. 1, pp. 60–65, 2003.

[146] A. Kaban, "Improved bounds on the dot product under random projection and random sign projection," in *KDD*, 2015.

[147] Q. Shi, C. Shen, R. Hill, and A. v. d. Hengel, "Is margin preserved after random projection?" *arXiv preprint arXiv:1206.4651*, 2012.

[148] J. A. Cuesta-Albertos, A. Cuevas, and R. Fraiman, "On projection-based tests for directional and compositional data," *Statistics and Computing*, vol. 19, no. 4, p. 367, 2009.

[149] R. J. Durrant and A. Kaban, "Random projections as regularizers: Learning a linear discriminant from fewer observations than dimensions," *Machine Learning*, 2015.

[150] E. P. Xing, M. I. Jordan, S. J. Russell, and A. Y. Ng, "Distance metric learning with application to clustering with side-information," in *NeurIPS*, 2003.

[151] R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin, "A simple proof of the restricted isometry property for random matrices," *Constructive Approximation*, 2008.

[152] Y. Plan and R. Vershynin, "One-bit compressed sensing by linear programming," *Communications on Pure and Applied Mathematics*, 2013.

[153] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on information theory*, 2006.

[154] T. Zhou and D. Tao, "Godec: Randomized low-rank & sparse matrix decomposition in noisy case," in *ICML*, 2011.

[155] N. Ailon and B. Chazelle, "Approximate nearest neighbors and the fast johnson-lindenstrauss transform," in *SOTC*, 2006.

[156] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theoretical Computer Science*, 2004.

[157] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Neural photo editing with introspective adversarial networks," *arXiv preprint arXiv:1609.07093*, 2016.

[158] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *CVPR*, 2017.

[159] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *NeurIPS*, 2017.

[160] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *CVPR*, 2015.

[161] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[162] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to sgd," *arXiv preprint arXiv:1712.07628*, 2017.

[163] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019.

[164] S. Gunasekar, B. E. Woodworth, S. Bhojanapalli, B. Neyshabur, and N. Srebro, "Implicit regularization in matrix factorization," in *NeurIPS*, 2017.

[165] S. Gunasekar, J. D. Lee, D. Soudry, and N. Srebro, "Implicit bias of gradient descent on linear convolutional networks," in *NeurIPS*, 2018.

[166] K. Kawaguchi, "Deep learning without poor local minima," in *NeurIPS*, 2016.

[167] Y. Li, T. Ma, and H. Zhang, "Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations," in *COLT*, 2018.

[168] X. Ding, Y. Guo, G. Ding, and J. Han, "Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks," in *ICCV*, 2019.

[169] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate o (1/k$\hat{2}$)," in *Dokl. akad. nauk Sssr*, 1983.

[170] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *JMLR*, 2011.

[171] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[172] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[173] M. D. Zeiler, "Adadelta: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[174] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang, "Deep fried convnets," in *ICCV*, 2015.

[175] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *BMVC*, 2014.

[176] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *CVPR*, 2015.

[177] M. Wang, B. Liu, and H. Foroosh, "Factorized convolutional neural networks," in *ICCV Workshops*, 2017.

[178] W. Hoffmann, "Iterative algorithms for gram-schmidt orthogonalization," *Computing*, vol. 41, no. 4, pp. 335–348, 1989.

[179] J. Li, F. Li, and S. Todorovic, "Efficient riemannian optimization on the stiefel manifold via the cayley transform," in *ICLR*, 2020.

[180] Z. Wen and W. Yin, "A feasible method for optimization with orthogonality constraints," *Mathematical Programming*, 2013.

[181] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas, "Full-capacity unitary recurrent neural networks," in *NeurIPS*, 2016.

[182] M. Lezcano-Casado and D. Martinez-Rubio, "Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group," *arXiv preprint arXiv:1901.08428*, 2019.

[183] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *ICML*, 2016.

[184] M. Henaff, A. Szlam, and Y. LeCun, "Recurrent orthogonal networks and long-memory tasks," *arXiv preprint arXiv:1602.06662*, 2016.

[185] L. Jing, Y. Shen, T. Dubcek, J. Peurifoy, S. Skirlo, Y. LeCun, M. Tegmark, and M. Soljacic, "Tunable efficient unitary neural networks (eunn) and their application to rnns," in *ICML*, 2017.

[186] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," *arXiv preprint arXiv:1811.03962*, 2018.

[187] S. S. Du, J. D. Lee, Y. Tian, B. Poczos, and A. Singh, "Gradient descent learns one-hidden-layer cnn: Don't be afraid of spurious local minima," *arXiv preprint arXiv:1712.00779*, 2017.

[188] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[189] W.-Y. Chen, Y.-C. Liu, Z. Kira, Y.-C. F. Wang, and J.-B. Huang, "A closer look at few-shot classification," *arXiv preprint arXiv:1904.04232*, 2019.

[190] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, 2008.

[191] Y. Nakatsukasa, "Eigenvalue perturbation bounds for hermitian block tridiagonal matrices," *Applied Numerical Mathematics*, vol. 62, no. 1, pp. 67–78, 2012.

[192] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," *arXiv preprint arXiv:1610.08401*, 2016.

[193] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *CVPR*, 2016.

[194] N. Papernot, N. Carlini, I. Goodfellow, R. Feinman, F. Faghri, A. Matyasko, K. Hambardzumyan, Y.-L. Juang, A. Kurakin, R. Sheatsley, *et al.*, "Cleverhans v2.0.0: An adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, 2016.

[195] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *NeurIPS*, 2017.

[196] D. Warde-Farley and Y. Bengio, "Improving generative adversarial networks with denoising feature matching," in *ICLR*, 2017.

[197] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[198] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao, "Ms-celeb-1m: A dataset and benchmark for large-scale face recognition," in *ECCV*, 2016.

[199] F. Wang, L. Chen, C. Li, S. Huang, Y. Chen, C. Qian, and C. C. Loy, "The devil of face recognition is in the noise," in *ECCV*, 2018.

[200] W. Liu, R. Lin, Z. Liu, L. Xiong, and L. Song, "Orthogonal over-parameterized training," *Technical Report*, 2020.

[201] J. A. Cuesta-Albertos, R. Fraiman, and T. Ransford, "A sharp form of the cramer–wold theorem," *Journal of Theoretical Probability*, vol. 20, no. 2, pp. 201–209, 2007.

[202] V. Srivastava, "A unified view of the orthogonalization methods," *Journal of Physics A: Mathematical and General*, vol. 33, no. 35, p. 6219, 2000.

[203] R. N. Annavarapu, "Singular value decomposition and the centrality of löwdin orthogonalizations," *American Journal of Computational and Applied Mathematics*, vol. 3, no. 1, pp. 33–35, 2013.

[204] D. Hardin and E. Saff, "Minimal riesz energy point configurations for rectifiable d-dimensional manifolds," *Advances in Mathematics*, vol. 193, no. 1, pp. 174–204, 2005.

[205] N. Landkof, *Foundations of modern potential theory*. Springer-Verlag, 1972.

[206] D. Soudry and Y. Carmon, "No bad local minima: Data independent training error guarantees for multilayer neural networks," *arXiv preprint arXiv:1605.08361*, 2016.

[207] J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht, "Gradient descent only converges to minimizers," in *COLT*, 2016.

[208] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv preprint arXiv:1609.09106*, 2016.

[209] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool, "Dynamic filter networks," in *NeurIPS*, 2016.

[210] A. Mallya, D. Davis, and S. Lazebnik, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *ECCV*, 2018.

[211] J. S. Brauchart, A. B. Reznikov, E. B. Saff, I. H. Sloan, Y. G. Wang, and R. S. Womersley, "Random point sets on the sphere—hole radii, covering, and separation," *Experimental Mathematics*, vol. 27, no. 1, pp. 62–81, 2018.

[212] A. Breger, M. Ehler, and M. Graf, "Points on manifolds with asymptotically optimal covering radius," *Journal of Complexity*, vol. 48, pp. 1–14, 2018.

[213] M. Reitzner, "Stochastical approximation of smooth convex bodies," *Mathematika*, vol. 51, no. 1-2, pp. 11–29, 2004.

[214] I. H. Sloan and R. S. Womersley, "Extremal systems of points and numerical integration on the sphere," *Advances in Computational Mathematics*, vol. 21, no. 1-2, pp. 107–125, 2004.